conference

proceedings

First Conference on Network Administration (NETA '99) Proceedings

Santa Clara, California
April 7–10, 1999

Sponsored by
The USENIX Association

USENIX®
The Advanced Computing Systems Association

USENIX First Conference on Network Administration (NETA '99) Proceedings Santa Clara, California April 1999

For additional copies of these proceedings contact:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Phone: 510 528 8649
FAX: 510 548 5738
Email: office@usenix.org
WWW URL: http://www.usenix.org

The price is $18 for members and $24 for nonmembers.
Outside the U.S.A. and Canada, please add
$11 per copy for postage (via air printed matter).

ISBN 1-880446-36-7

Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

Proceedings of the

First Conference on Network Administration

(NETA '99)

April 7–10, 1999
Santa Clara, California, USA

Program Committee

Program Co-Chairs

David Williamson, *Global Networking and Computing*
Paul Ebersman, *Vixie Enterprises*

Program Committee

Fuat Baran, *Columbia University*
Brent Chapman, *Covad Communications Company*
Phil Draughon, *RMS Business Systems*
Paul Ferguson, *Cisco Systems*
Jeff Jensen, *WebTV Networks, Inc.*
Bill LeFebvre, *Group sys Consulting*
Bryan McDonald, *Global Networking and Computing*
Hal Pomeranz, *Deer Run Associates*
Wade Warner, *Georgia State University*

USENIX Association

Ellie Young, *Executive Director*
Judith F. DesHarnais, *Conference Coordinator*
Toni Veglia, *Member Services Coordinator*
Cynthia Deno, *Marketing Director*
Linda Barnett, *Marketing and Communications Coordinator*
Jane-Ellen Long, *Publications Director*
Daniel V. Klein, *Tutorials Coordinator*

Contents

First Conference on Network Administration (NETA '99)

April 7–10, 1999
Santa Clara, California, USA

Message from the Program Chairs

NETA '99 is the first in an annual series of conferences expressly for the networking world. In this inaugural year, we have an exciting program of invited talks and refereed papers, with something for everyone. Our goal for this year's event was to create an environment for discussions of all facets of network administration.

In reaching this goal, the program committee has been an invaluable resource. Without their hard work in reviewing papers and shaping the final program, this conference wouldn't exist. The combined experience of the committee members, combined with the quality of submissions, has allowed us to produce an amazing range of presentations.

We'd like to extend our sincerest thanks to all of the authors who submitted abstracts for papers and talks. Although we couldn't accept all of the submissions, the quality was astonishingly high for the first year of an unknown conference. All paper authors are highly encouraged to submit again next year, as that program committee will again be looking for the best papers available.

Special thanks go to our bosses for their support throughout the process of the creation of this conference, from conception to completion.

Finally, a big thank you goes to Ellie Young and the USENIX office. Without the talent and organization of her amazing crew, no USENIX or SAGE conference would ever occur. If you see one of them at the conference, give them a pat on the back for all of their hard work.

Again, thanks for attending NETA '99!

David Williamson Paul Ebersman
Program Co-chairs

Driving by the Rear-View Mirror: Managing a Network with Cricket

Jeff R. Allen
WebTV Networks, Inc.

Abstract

Cricket is a tool that lets users visualize a set of measurements over time. It was designed to assist network administrators by letting them see and respond to patterns in their network. In this paper, I will describe the need we saw and attempted to resolve by writing Cricket, then describe the solution we came up with. Finally, I will describe some future work we expect to do to make Cricket a more proactive monitoring tool.

The Need

When running a complicated network, there are a multitude of things one needs to keep an eye on. Clearly, immediate concerns like connectivity, link status, and routing stability are top in most administrator's minds. Long-term issues like architecture and technology decisions float in the back of our minds. Often, that leaves no room for the most interesting questions, which help with both short term issues and long term ones. Some of the questions we found ourselves asking about various network components were:

- What is the current state of a component?
- What has it been recently?
- What will it likely be in 5 minutes? In an hour? Is that what we expect it to be?
- What long-term trends can we discern?

Taking the time to ask these questions allows us to step up a level in our monitoring, and move from a reactive realm (i.e. Is the link to Europe up?) to a more contemplative and predictive realm (i.e. Are we seeing an unexpected burst of traffic to Europe?) It's obviously not possible to definitively say that this makes our job easier and our network run more smoothly, but it certainly seems that way. Simply having the data available to be able to say "things look OK", gives us a peace of mind that simply reacting to the network would never give us.

Figure 1 shows a screenshot from Cricket, showing the normal way that data is displayed about a target. In this case, we are looking at the traffic on one of our OC3's, which was operating normally until an outage started a little after noon. From this same view, we can tell that in the last week, the peak bandwidth in use on this link was 60 Mb/sec, on Wednesday evening. By glancing at one page, we can answer many of the questions posed above. By looking at longer time scales (accessed via the links in the upper right) we could answer still other questions about the long-term behavior of the traffic on this link.

One of the reasons it's hard to monitor a network is that there are a lot of different components, each with different operating characteristics and monitoring needs. For instance, in our network, some of the components that we use Cricket to monitor are:

- WAN and LAN interfaces on routers
- Router operating state (memory, temperature)
- Switch (or hub) port bandwidth
- Rack-mounted modem usage
- DNS activity
- Host Components (disk, load average, swap)

Note that we monitor some things (DNS, hosts) that are not directly related to the health of the physical network. In our shop, the "host folks" and the "network folks" work together very closely. There is no artificial separation between the two. I will focus on monitoring the networking components, however, keep in mind that Cricket is capable of monitoring many types of components.

Just as the components we wish to monitor are varied, so are the ways in which we talk to them. Sometimes, we can simply talk SNMP, then either record the data directly or do a bit of post-processing to derive a rate of some kind. Other times, we fetch data via SNMP, then post-process it in some more complicated way to get a final data point. For instance, to monitor modem bank usage, we fetch the current state of all the modems, and count the number that are off-hook. Sometimes we simply run a shell command on another data-gathering system. For instance, we can use Unix tools like 'wc' or simple Perl scripts to derive a data point from the data already collected by syslogd. Finally, sometimes we want to measure and observe an

webtv

Graphs for uunet-oc3-2

Summary

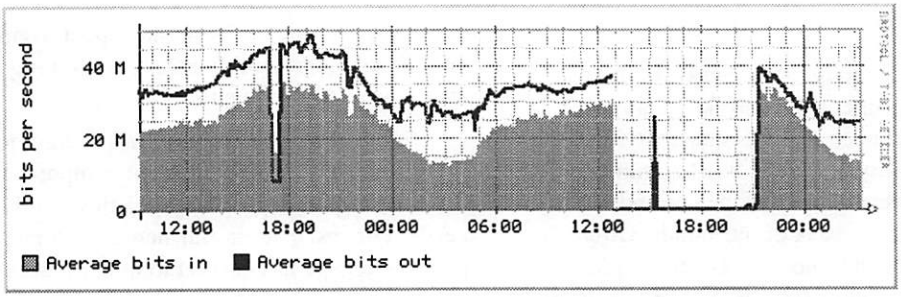
OC3 to UUNet in San Jose

Values at last update:
Average bits in: 13 Mbs [?] Average bits out: 24 Mbs [?]
Last updated at Thu Feb 18 03:11:03 1999

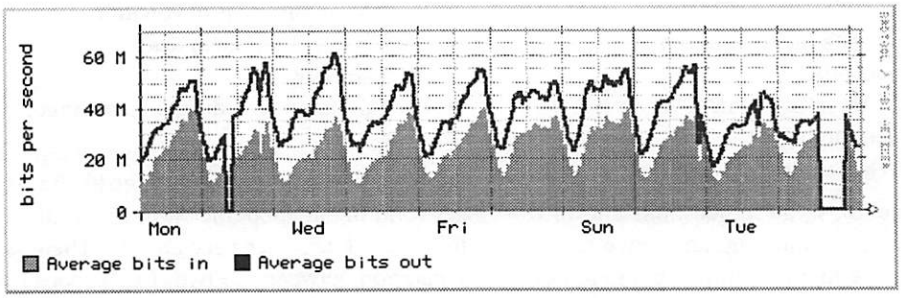
Time Ranges:

Hourly
Daily
Short-Term
Long-Term
All

Hourly graph



Daily graph



About the data...

Average bits in
The rate of input in octets.

Average bits out
The rate of output in octets.

Cricket
Version 0.59

For questions or comments about this data, contact the Cricket Admins.

Built on Tobi Oetiker's RRDTOOL

Figure 1. An example of Cricket in action. These graphs show an OC3 outage.

aggregate of other data sources. In a site with multiple Internet links, it would be interesting, for instance, to plot the total Internet bandwidth across all links.

MRTG to the Rescue

Around the time we[1] were bringing up the WebTV network, we fetched and installed a tool called MRTG[2] to let us start answering the kinds of questions mentioned above about our WAN links. In fact, MRTG was in use at WebTV Networks before we had brought in any commercial network management tools at all. It was only by using, and coming to depend on the MRTG graphs, that we were able to take a step back and figure out why they were so useful to us. After getting ourselves thoroughly addicted to MRTG, even while attempting to roll out another network management system, we were forced to ask the question, "What is it that MRTG does that we can't get elsewhere." The answer to that question makes up the bulk of the first section: MRTG let us manage our network better than reactive systems.

So, there we were, addicted to MRTG, using it in new situations and in ways it was never designed to be used. MRTG started showing signs that it would not scale to handle the new jobs we wanted to throw at it. Something needed to be done, and quickly. A true addict will not wait patiently for a new drug!

Before we threw out MRTG and started over, we thought it might be a good idea to figure out what it does right, and make certain that whatever we got to replace MRTG could at least do those things. Each view of the data has just enough detail to tell the story, but not enough to hide the good bits. For instance, the weekly views show today and this day last week. There is an enforced data density so that graphs have just enough data to tell an accurate story, but not enough to overwhelm the viewer. In addition, the auto-scaling feature consistently produces readable graphs (though there is a bit of room for improvement on this front). It uses a fixed size database which grows linearly with the number of monitored devices, not time. It needs no extra "cleanup" work. Because MRTG is web-based, we have a platform independent tool: all it takes is a web browser to check the status of the network. You can even use a WebTV®-based Internet Unit to see graphs of your network on your home television!

Perhaps more important than all that, though, MRTG passes the "can it solve real problems" test:

- It shows BGP failover very well.
- It is an invaluable tool in traffic shifting/balancing.
- It lets us answer the question "How did it look before it broke?"

To see how we use graphs to identify and understand real problems, examine figures 2 and 3. These two graphs are selected snapshots of the graphs available to us during backbone routing instability on one of our peer's networks. During this event, it's clear that about 10 Mb/sec of traffic shifted over to the second peer. We were able to use Cricket to find the nature of the problem, see that the outbound traffic had failed over to non-optimal, but functional link, and finally verify that the traffic returned to normal after the event.

Finally, Cricket passes the "manager test" with flying colors. Managers immediately understand the need for a bandwidth upgrade once they see a graph produced with MRTG. Even more astoundingly, it even passes the "executive test", which is very important since executives tend to need to sign orders for OC3's and above. The simple web-based user interface makes it easy to print and share the graphs with management, making it easier for everyone to do their job.

Regardless of all these great features, we identified problems we wanted to address in the next generation system. First, MRTG has a narrow view of the world: all targets have exactly two data sources (used by MRTG for bandwidth in and bandwidth out). We wanted to be able to measure other things about components. MRTG also has severe performance problems, resulting from the way it reads and writes data. MRTG was incrementally developed and the performance problem only became apparent after it was too late to fix it. Usually, it's not an issue, because MRTG is mostly used in small installations. However, with all the different ways we were trying to use MRTG, performance became a problem for us.

There are ways to overcome these performance problems by arranging to run multiple instances of MRTG in parallel. However, the configuration needed to run it in parallel is even more complicated than the usual configuration. For our varied uses, the configuration was already unwieldy and inefficient. The last thing we needed was to make it even more complicated!

[1]Actually, this was before I arrived at WebTV Networks. Joe Balboa originally brought MRTG to our company.

[2]More information about MRTG is available from **http://www.mrtg.org**.

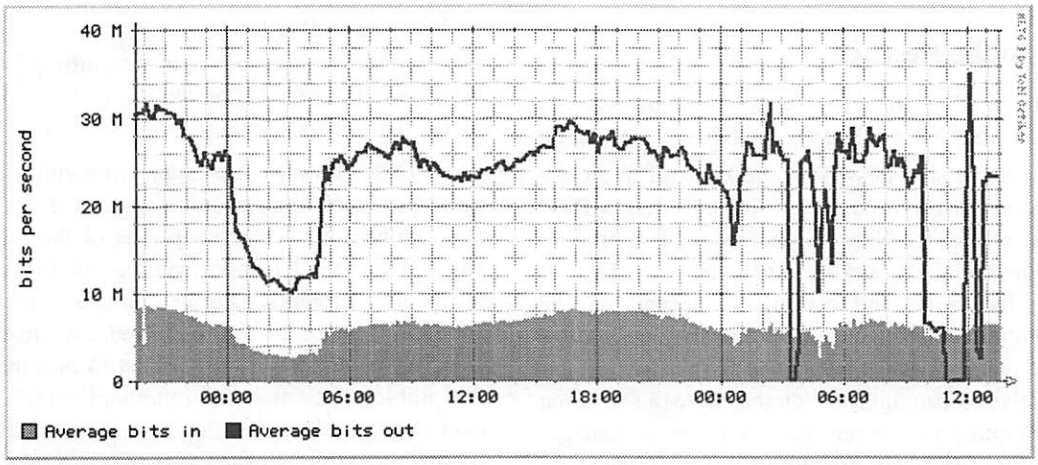


Figure 2. Traffic on a Fast Ethernet handoff to a peer of ours. Their backbone was suffering routing instability at the time.

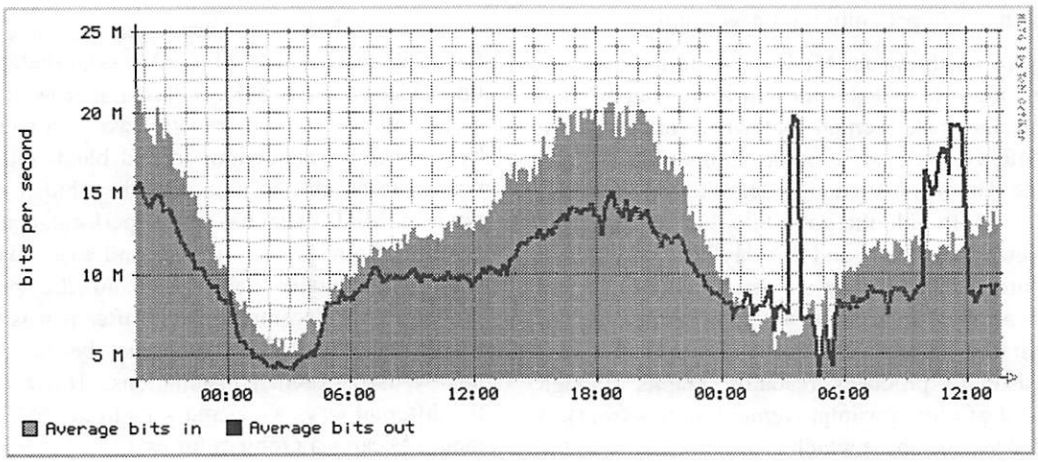


Figure 3. Traffic on a DS3 to another peer during the routing instability shown in Figure 2.

Why Not Commercial Tools?

Some might recommend commercial products as a replacement for MRTG, since commercial products are supposed to be designed to scale well and be easy to configure. Our experience is that commercial tools introduce as many problems as they purport to solve, and often leave the original problems (scalable performance and configuration) unsolved. Here's an undoubtedly one-sided list of the problems inherent in most commercial tools that could have helped us.

First and foremost is cost. It's expensive to make good software. No one will argue that point. In fact, it has been more expensive than we expected to develop Cricket. But when buying an expensive network management tool, things get ugly. It takes too much time, paperwork, patience, and persuading management, even if you do have the budget for a commercial tool. Evaluating and rolling out a free software package can happen in an afternoon, if you put your mind to it. There's simply no downside risk – install it and if it solves your problem, keep it. If it doesn't meet your expectations, delete it and move on. You get what you pay for, but sometimes it's what you are not getting (support hassles and licensing nightmares) that really counts.

With commercial tools, database formats are often secret, API's (usually only available at extra cost) are either a pain to work with, or are major performance pigs. Customer support varies, but the simple fact is that even stellar customer support won't do for you what getting your own two hands dirty in the code will do: give you a deep understanding of your NMS and the confidence to trust the system and your ability to operate it. Then, there's the version tango. The NMS doesn't find out about new router models until 6 months after you've already deployed them because the router company and the NMS company are feuding. The trouble-tracking system you are using is only certified to talk to the next version of the NMS that you cannot use yet because that version introduces a critical bug that no other customers are seeing, and thus, will not get repaired until you threaten to revoke your support license – which, by the way, does not expire until next December.

Finally, let's be honest: Free Software rules. Who would you rather trust with the smooth operation of your network, a bunch of folks who are in the same boat as you, or a giant vendor with 100-page implementation plans and slick salesmen? The choice is obvious, to me at least: network managers design better tools to do their job.[3] Sometimes commercial tools are the right tool for the job, but in this case, it made more sense to go at it on our own. It will make even more sense for your organization, since WebTV Networks has done much of the hard work. All you need to do is download, install and configure Cricket.

The Solution

The solution comes as two pieces, a new database and a new user-visible front-end. By changing out the back-end, we addressed the flexibility and performance goals. By changing out the front-end, we improved the user interface, both for configuring the system and viewing the graphs. The two pieces are developed semi-independently, one by me (in California) and one by the original author of MRTG, Tobias Oetiker (in Switzerland). Keeping the two pieces separate makes it easier to develop them, and it also means the database component can be reused in other contexts. RRD, the database backend, is distributed separately from Cricket[4].

We are deeply indebted to Tobias for his hard work on the database back-end. Without it, Cricket would be nothing. By engineering RRD correctly from the start, and by improving it's reliability over the last year, Tobias solved over half our problems with MRTG himself. I simply had the easy job of passing the right data into his code. All the seeming magic of Cricket comes from Tobias's RRD code.

RRD: The Round Robin Database

The next generation back-end is called RRD, the Round Robin Database. It takes over exactly the same jobs the back end provided with MRTG did: storing and rolling up the data and generating graphs from the stored data. RRD is written in C, and comes in both a Perl module and a command line version, which can be used interactively or across a pipe from scripts. It can be used either directly by simple scripts, or via frontends like Cricket. Other frontends are available from the RRD website.

RRD achieves high performance by using binary files to minimize I/O during the common update operation. The "round robin" in RRD refers to the fact that

[3]Writing code is another issue entirely. As a toolsmith, my job is to write the tools WebTV employees need to do their work. Much as they might like to, the network administrators at WebTV Networks don't normally have the time to hack the Cricket code.

[4]The RRD website is at:
http://ee-staff.ethz.ch/~oetiker/webtools/rrdtool.

RRD uses circular buffers to minimize I/O. RRD is at least one, and possibly two orders of magnitude faster than MRTG's backend, depending on how you measure it. It's truly incredible to watch it chew through data if you have ever seen MRTG trying to handle the same job.

RRD is more flexible than MRTG in at least two dimensions. First, it can take data from an arbitrary number of data sources. MRTG was limited to two datasources. Originally they were reserved for input bandwidth and output bandwidth, though they got co-opted for other measurements by many MRTG hackers. RRD can also keep an arbitrary number of data arrays, each fed at a different rate. For instance, you can keep 600 samples taken every 5 minutes alongside 600 samples taken every 30 minutes. Thus, you can have 5 minute data stretching back 50 hours into the past, and 30 minute data stretching back 12 days. When you draw a graph of this data, you can see recent data in high resolution, and older data in lower resolution. This is one of the original features of MRTG that made us so happy. In RRD this feature is completely configurable.

RRD stores data with higher precision than ever before (float versus integer) which allows us to measure bigger things (OC3's) and smaller things (Unix load averages) without relying on goofy scaling hacks, as we were required to do with MRTG. The data file where this data lives on disk is still a fixed size over time, and scales with the product of the number of datasources and number and sizes of the data arrays. The data storage in use for an entire installation still scales linearly with the number of targets under observation. At WebTV Networks, we have Cricket configured to store 6 variables for every router and switch interface. Each interface requires 174 kilobytes on disk, which is enough room to store all the data for 600 days. Of course, your mileage may vary; Cricket is configurable, so you can choose to keep more or less data, depending on your needs.

RRD still provides the same simple, sparse graphs we grew to love while using MRTG. There is now much more flexibility at the time the graph is generated. For instance, we can choose to show some data sources, but not others. We can choose to scale a data source using an arbitrary mathematical expression. For instance, we can fetch the temperature from a router in Celsius, and present it to the user in Fahrenheit. Since almost all data seems to arrive in exactly the wrong units, it's quite helpful to have this flexibility. Do you think about Ethernet capacity in "megabytes/sec"? The scaling feature lets us turn that measurement into "megabits/sec". It is also possible to integrate data from multiple sources into a single graph. For instance, you could make a summary graph showing the sum of the traffic across all of your Internet links.

The Config Tree

Recall that part of the problem was flexibility and performance of the database, both of which RRD solved. The other part of the problem was scalability of the configuration. Our solution is something called the config tree. A config tree is a set of configuration files organized into a tree. This hierarchical configuration structure allows us to use inheritance to reduce repeated information in the configuration. To simplify implementation of the config tree, the inheritance strictly follows the directory structure; complicated concepts like multiple inheritance are not supported. The attributes that are present in a leaf node are the union of all the attributes in all the nodes on a path leading from the root of the config tree to the leaf node. Lower nodes can override higher nodes, so it's possible to make exceptions in certain subtrees, and do other clever sleight of hand.

It's easier to understand the config tree by looking at an example (see figure 4). Attributes that all of the system will share are located at the root of the config tree. For instance, the length of the polling interval is set there. At the next level, we set attributes that will be restricted to the current subtree. At this level, typically you will find the target type. Finally at the lowest level we set things that will vary on a per-target basis. For instance, we set the interface name that we are trying to measure here. By using the power of inheritance, you can avoid repeating the information at the top of

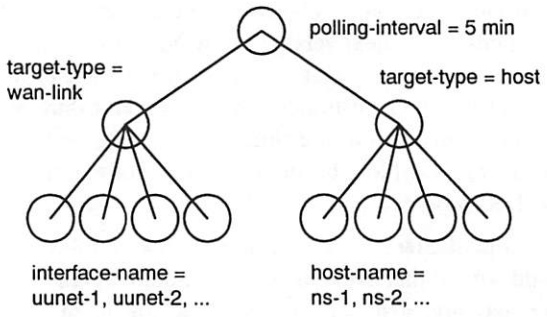


Figure 4. A simplified example of a config tree.

the config tree many times near the bottom of the config tree. The three level config tree in the example is the simplest in common use. The one that ships with Cricket in the sample-config directory works like this. Config trees can and do have many more levels. At WebTV Networks, for instance, we add levels to break apart routers in different data centers to make the directory listings more manageable. There are no built-in limits on the shape of the config tree, only practical ones.

Figure 5 shows the power of the string expansion feature in Cricket. When it comes across strings in the config tree with special markers in them, it expands the strings in much the same way a scripting language expands variables. The sample config tree that ships with Cricket uses this feature in several places to dynamically build strings from settings already available. In the example, the short and long descriptions for the target are set based on some data inherited from high in the tree, and other data related to the target itself. In order to make this feature even more useful, we introduced auto-variables, which are set to certain strings automatically by Cricket. They are available for use by expansions, making it possible to automatically generate things that change for every target, like the data file location. The same expansion system is currently used to make it possible to customize the HTML output of the user interface component, though it is still not yet as flexible as we would like it to be.

Cricket's collector is single-threaded, and thus spends a fair amount of time waiting for data to arrive over the network. The wall-clock run time for each instance of the collector is limited to the length of the desired polling interval (or else you might miss polling cycles, which would be unfortunate). Thus, the number of targets that can be polled by a single collector on a single host is limited. To get maximum polling capacity, it is necessary to run several collectors in parallel, each working on a different subset of the total set of targets to be polled. This was hardly a surprise to us, since we needed to use the same technique to boost the performance of our MRTG installation. In fact, we designed the config tree to make partitioning the set of targets easy, so parallelizing Cricket is simply a matter of starting the right collectors at the right time, each operating on the right subtrees of the global config tree. There is a wrapper script for the collector which does this, as well as other housekeeping chores like managing logfiles, and checking them for errors.

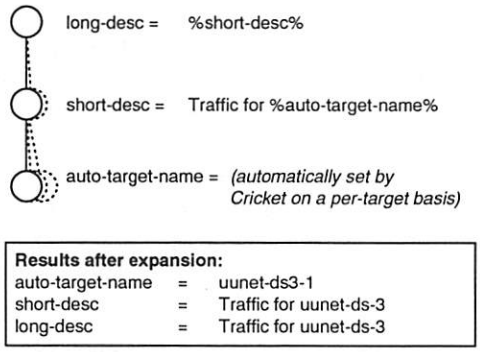


Figure 5. An example of variable expansion.

The config tree has several other advantages we stumbled upon after using it for a while. Because the subtrees are mostly independent from one another, multiple administrators can work on the config tree without impacting each other. The hierarchical structure makes it simple to make standards that apply across all the different applications. It also makes it possible, when necessary, to make exceptions for certain devices. Of course, in a perfect world, this wouldn't be necessary. But this isn't a tool for a perfect world, it's a tool that was written to solve problems in the real world. Sometimes it just happens that we need to query one device with different OID's, or that the community name is different on a certain device. At the same time, though, a properly designed config tree can save you a lot of work when you decide to make some kind of widespread change to the system. Instead of changing the data everywhere, you can change it in one place.

Contemplating our fully populated config tree gave us another idea. One of the hardest parts of maintaining a suite of network management products is trying to keep their configurations up to date and in sync with one another. What if we declared, by fiat, that our Cricket config tree was the one true source of configuration information, and that all others should be derived from Cricket's config tree? We decided to design the config tree to be a simple collection of dictionaries. As the config tree is read the parser pays no attention to what keys and values it finds. It simply stores them away. Later, the application which called the parser (Cricket in this case) does lookups in the dictionary looking for data it understands which will control its behavior. Since it is doing lookups into the data, it never even sees data that it's not interested in. In theory, lots of different tools could all rely on the config tree, leveraging the investment made into creating and

updating a config tree. In practice, a fair amount of code that belongs in the parser ended up in the Cricket application itself, which means that it will take some reengineering to make the dream of the One True Config Tree come true.

The User Interface

The fastest, most flexible data collection on the planet doesn't help at all if there's no way to see the data. MRTG led us down the right path here; the graphs it makes are simple and useful, and they are presented in a web page viewable on virtually any platform. For Cricket, we decided to make the HTML and the graphs on the fly, instead of caching completed graphs like MRTG does. This saved some of the processing overhead inherent in MRTG's regular polling cycle. As a bonus, using the CGI script instead of pre-generated HTML soundly defeats the over-aggressive browser-side caching that plagues many MRTG installations. On the down side, the startup time for the CGI script can be a problem, as can the graph generation time, both of which the user perceives as lag in the user interface. The startup time problems could be mitigated by the mod_perl Apache module, and Cricket already uses a graph caching scheme to attempt to avoid wasting time re-rendering graphs. It's generally true, though, that the Cricket user interface is slower than the MRTG interface.

The CGI graph browser was specifically designed to maximize flexibility. It is read-only, which keeps security concerns to a minimum. If access control to sensitive information is required, it is currently only possible on a per-installation basis. This is one drawback to the CGI graph browser, but it has not been a problem for us. There are also ways to run a graph browser on a subset of all the collected data, so access controls could probably be implemented by a determined Cricket user. The graph browser operates entirely on URL's which can be incorporated into web pages outside the Cricket system, or stored as bookmarks. This has made it possible to build "views" of the Cricket data which are annotated in various ways to assist operations personnel. We can also use these external pages to collect a set of interesting graphs which might be difficult to navigate among inside the graph browser.

To aid in analyzing the data, it can be useful to assemble graphs out of various data streams which are separately collected and stored by Cricket. So far, this capability is limited to putting multiple graphs on one page, and gathering data from several sources and plotting it in summary on a single graph. These advanced features intended to make it easier to look at the data are a tough sell; they tend to be difficult to implement within the current design, and are only used when trying to solve a particular kind of problem. Of course, having the right tool for the job is invaluable, so it's tempting to try to make Cricket do everything under the sun.

While Cricket is the right tool for these analysis jobs, it's not yet infinitely flexible. It presents useful data that answers 90% of the questions, but the jury's still out on the costs and benefits of adding the complexity necessary to answer the last 9% of the questions. What about the last 1%? We'll never get there, and we don't plan to try. Sometimes, the data suggests questions that are better answered with other tools. The trick is knowing when to turn elsewhere to get your answer.

Instance Mapping

MRTG relies on SNMP instance numbers to identify which of the interfaces on a given router it will pick up data from. One of the things we knew we hated about MRTG was the necessity of re-adjusting instance numbers after any router configuration change. It was a small, but persistent, pain in the neck. It's a dreadfully mundane job, and especially annoying on the large routers we use, which have lots of interfaces. Still, it didn't happen that often, and it hardly showed up on the radar screen of "things we want to improve in MRTG".

To be fair, the instance number problem is not even really MRTG's fault. It's due to an annoying compromise in the SNMP specification that lets network devices get away with murder in the interest of making them easier to implement. This was done under the assumption that polling systems will be smarter than network devices, and can make up for the lack of intelligence in the managed nodes. SNMP devices are allowed to renumber their interfaces behind the poller's back potentially between every single polling cycle. What's a poller to do? The system can either force humans to fix things up when necessary, or it can attempt to figure out the right instance number on it's own. MRTG uses the former strategy, while Cricket has a feature called "instance mapping" to implement the latter strategy.

Cricket's instance mapping feature allows administrators to configure a target by name, and then let Cricket do the necessary SNMP lookups to map that name into an instance number. From then on, it's Cricket's job to keep the instance number correct, no matter how hard the device tries to confuse it. Cricket does this magic in a bulletproof but efficient way. To

begin with, it uses a series of SNMP GET-NEXT operations (i.e. a table walk) to map the instance name to a number. Once it has a valid instance number, it caches it. The next time it uses the cached instance number, it fetches the name again, along with the polled data. If a "no such instance" error is returned, or the name no longer matches, Cricket discards the polled data, maps the name to a new instance number, and fetches the data again. The new instance number is cached, and will be used until the next re-mapping is required. Fetching the name every time like this amounts to a small overhead in the polling transaction, but it ensures that re-mapping happens as soon as necessary, without any human interaction. Presumably commercial systems solve this problem the same way.

It turns out that this is not just a nice feature, it's a required one in some cases. It was designed to be extremely flexible, since I hope to use Cricket to monitor items in the Host MIB which tend to get different instances all the time (disks and processes, among others). To this end, Cricket can even match against names which are regular expressions, making it easy to find a given process, no matter how it was invoked. I have also been told that the virtual interfaces created and destroyed by Cisco's ATM LANE implementation come and go with unpredictable names, making them very hard to monitor with conventional tools. A Cricket user managed to monitor these virtual interfaces for the first time ever using the instance mapping code and regular expression matching. His exact comment on the triumph was "Instance mapping is the best thing since sliced bread!"

Future Directions

Our hope for Cricket is that it finds a place in other operations centers, and that it helps improve customer service all around. Poor service reflects poorly on all of us, and it's in the industry's best interests to develop useful tools to raise Quality of Service.

The major features we needed from Cricket are in place now, and it has superseded MRTG at WebTV Networks. The bulk of the work is done. However, there is the usual laundry list of "todo" items for Cricket. We will do ongoing work on it, especially to integrate patches from interested helpers.

As we have come to depend on graphs to tell us about the state of the WebTV Service, we have stumbled across a principle that should have been obvious to start with. Simply put, lots of graphs are not a proactive tool. It's not reasonable to expect humans to continually review each of the over 4000 graphs we can produce and watch for anomalies. Of course, Cricket was never intended to simply find links that are down, or hosts that are crashed. There are other tools that we already use for that kind of monitoring. Subtle problems in the system manifest themselves as subtle changes in the patterns on the graph. It is these problems that we seek to discover when we humans browse the graphs. We are looking for graphs that don't "look right".

The obvious solution to the problem of too many graphs is to ask Cricket itself to evaluate the graphs and flag those that don't "look right" for further analysis by a human. This is about the time things get really complicated; we could apply all the fancy pattern recognition tools that the computer science community has to offer, including neural networks, signal processing, and fuzzy logic. We took a more pragmatic view and asked the question, "How do we humans know when it looks right?" The answer in all cases was that we compare the current behavior to past behavior. Now the problem looks a bit more solvable. We want to create a system of thresholds which are set differently at different times of day. These thresholds will be derived from past data, so that deviation from past patterns results in violated thresholds, which in turn results in an alert that a human sees. With an appropriate GUI, administrators could adjust the suggested thresholds to avoid false positives. They could also mark known anomalous data with the GUI, so that future generated thresholds would not be affected by the "bad" data.

We have asked a group of students from Harvey Mudd College to work on this problem as part of the Computer Science Clinic program. The Clinic program offers a team of undergraduates a chance to work on a real-world problem, while offering companies like WebTV Networks access to bright students who can pursue a project independently. I work less than an hour a week with the team as a liaison to help them understand the problem and our expectations. In return, they are responsible for managing themselves and delivering a finished product at the end of the school year.

The threshold system the clinic team is working on will store its data in the config tree. It will take advantage of inheritance to eliminate duplicated thresholds. It will have a GUI that can be used to look at and modify the suggested time-based threshold curves which are generated using historical data. Threshold violations that are discovered by the collector as it fetches the data will either spawn a shell script, or forward the alert to an alert management system via SNMP traps. The result of the project will be incorporated into the standard Cricket distribution later this year. It will be available under the same license as Cricket itself, the GNU Public License (GPL).

Of course, the real future for Cricket lies with all of the other folks who pick it up and use it. Because it is distributed in source form, and is protected by the GPL, Cricket users are guaranteed the right to hack on it however they see fit. If you need Cricket to do something it cannot already do, you can write the code and share it as contributed software or as a patch. I'm looking forward to hearing from Cricket users about how it's working for them, and seeing what features they need to add to make it work even better.

Availability

The Cricket homepage is at:
http://www.munitions.com/~jra/cricket.

There's more information there, including the Cricket distribution, where to get the other things you need to make Cricket work, and what mailing lists you might want to join for help. People who want to get in touch with me personally can send e-mail to *jra@corp.webtv.net*.

Acknowledgments

Cricket would not have been written if WebTV Networks had never gotten so addicted to MRTG. So hats off to Tobias Oetiker for a great tool, and for coming through with RRD, a perfect encore. Lots of the ideas for how to improve Cricket came from Jeff Jensen, one of the network administrators at WebTV Networks. The WebTV Networks management lets me spend time on this project, and chose to make it freely available on the net, to boot. Laura de Leon reviewed this paper and helped me spot some unanswered questions. Thanks to you all for your help.

Don't Just Talk About The Weather - Manage it! A System for Measuring, Monitoring, and Managing Internet Performance and Connectivity

Cindy Bickerstaff, Ken True, Charles Smothers, Tod Oace, Jeff Sedayao
Intel Corporation
Clinton Wong
@Home Corporation

Abstract

In an environment where Internet access is mission-critical, Intel has created the Internet Measurement and Control System (IMCS) with three objectives: 1) Devise quantitative measures of Internet performance; 2) Monitor those metrics to detect performance problems before customers and employees start calling; and 3) Enable first line support in the Network Operations Center (NOC) to handle as many problems as possible without having to escalate to network engineering staff. Intel implements IMCS by measuring key statistics of ping measurements, HTTP GETs, and router accounting tables. Boundary conditions are set up for the key statistics, and alerts are sent if those conditions are exceeded. The NOC personnel that receive the alerts use predefined scripts for each kind of alert. To make IMCS accessible to all and very usable, IMCS presents all of its information on the Web. Even network debugging tools like ping and traceroute are accessible through web interfaces. IMCS has proven successful in detecting problems and changes in the Internet infrastructure, although problems have been encountered because of IMCS's active measurement techniques. Future improvements to IMCS include fixing the configuration format of boundary condition definitions, adding more services to be monitored, increasing the use of passive measurements, and improving how alerts are reported.

1. Introduction

Internet Performance is like the weather - people talk about it but feel helpless to do anything about it. But you can do something about it! Like many large multinational corporations, Intel has many thousands of employees and contractors who use the Internet through Internet gateways dispersed through the US and the world. Despite a massive investment in Internet Connectivity, Intel had no quantitative measurement of whether its Internet connections were performing well or poorly. In addition, Intel has a Network Operations Center manned 24x7 but only a limited number of engineers working on Internet Connectivity issues. In this environment of heavy use, no quantitative performance data, and small amount of staff, Intel created the Internet Measurement and Control System (IMCS) with the following objectives:

- Devise quantitative measures of Internet performance
- Automatically monitor those metrics to detect performance problems before customers and employees start calling
- Enable first line support in the Network Operations Center to handle as many problems as possible without having to escalate to the Internet network engineering staff

This paper describes how Intel built IMCS to meet these goals and how well IMCS can help in managing Internet performance. The first section details the challenges that lead us to create IMCS. It talks about Intel's Internet Connectivity environment and the key goals of IMCS. The second section goes over the IMCS approach - the techniques used to quantitatively measure Internet performance and our strategy to present that information and make it usable. The third section details how we implemented IMCS. It talks about specifically about our performance measurements, measurement architecture, and user interface design and implementation. The next section talks about Intel's experiences with IMCS, and the last section talks about our plans to enhance IMCS.

2. The Challenge of Internet Performance Management

Prior to the initial deployment of the IMCS in mid-1997, Intel's Internet use focussed on the use of Intel's corporate presence server (CPS) by external customers and on the use by employees for business use and reasonable personal use. The availability of the Internet had become a critical part of standard business proce-

dures much like a fax or a telephone. Since IMCS has been deployed, the criticality of the Internet has entered the Intel business processes for more than a billion dollars per month at our order placement eCommerce site. [1] These business procedures extend across all of Intel's customers, groups and divisions around the world. Multiple geographically diverse gateways have been deployed across Intel. Consistent access policies to these gateways' routers, servers and bastion hosts have been implemented. A tiered and scripted support approach was developed to address the requirements for 24x7x365 coverage, restricted secure access and limited Internet Connectivity staff. Skilled NOC staff were already available 24x7x365 for Intel network business needs and were the designated resources for first level support for Intel's Internet connectivity.

Now that the Internet has become a critical business tool, the following IMCS goals were established to meet the challenge of managing its performance:

- Define quantitative measures of Internet performance that can be used to manage connected ISP performance and track/act on levels of service delivered to external and internal customers.
- Detect performance problems before customers start calling to complain of performance problems
- Enable first line support to consistently and securely resolve as many problems as possible.

3. The IMCS Approach

Intel's approach to managing Internet performance is fairly straightforward. First, we want to quantitatively measure Internet performance. To do that, we define quantitative performance metrics corresponding to real performance issues and then consistently measure them. Once we have defined metrics and measure them, we monitor Intel's Internet gateways for when the measured values of the metrics violate some boundary condition for good performance. When a boundary violation is detected, the Intel NOC goes through carefully scripted actions in order to find the problem and contact our ISPs if necessary.

It's worth discussing this quantitative approach in a little more detail. The core of this approach is to obtain performance data, process it with an algorithm that includes comparison to usual performance and act when unusual performance is detected. Action takes place once the data demonstrates unusual behavior. Note that all parts of this process are critical. Actions are based on quantitative data. The data should be meaningful, reflecting in some way real Internet conditions. Comparison to usual performance or "limits" is important. We need a defined trigger for taking actions when a performance variable exceeds a limit. Otherwise, we may take actions when conditions do not warrant it (false positive) or not do anything when conditions demand it (false negative). Finally, actions are important. There is no point in measuring something if you are not going to act on the data.

The second part of Intel's approach involves the way IMCS presents information. All of IMCS is accessible from the Web by everyone at Intel. We do not hide our performance information since Internet performance affects almost everyone at Intel. All of the debugging tools like traceroute and ping and the troubleshooting scripts are made available to Intel's NOC (and everyone else) via the Web. Again, we do not hide performance behind some management console available only to a few people. This gives us incentive to do a good job with Internet performance. It also can be a convenient time saver. When there are questions about Internet performance, we have the ability to answer those questions with a URL showing the actual performance.

As part of the web display of performance, we graph our performance variables. This makes its easier for people to understand, rather than presenting raw data. There are a number of advantages to making our Internet debugging tools web based. Since they are web-based, our NOC and other interested parties do not need log in access to routers in order to debug problems. Also, the complexity of running certain debugging tools is reduced by allow users to execute the tools by filling in web forms and hitting a button.

4. Implementing IMCS

Now that we have described our approach to Internet performance monitoring, we will go over how we implemented it with IMCS. The first part of this section covers how we defined summary statistics for performance, what algorithms we used to determine whether boundary conditions were violated, and our statistical treatment of network performance data. The second part describes how we implement the user interface to IMCS – the alerting and web interfaces. The last part goes over some other implementation highlights such as implementation languages used and configuration interface design.

4.1. Creating Quantitative Measures of Internet Performance and Setting Boundary Conditions

As we mentioned in the discussion of our approach, IMCS measures various network performance metrics and trigger alarms when the data exceeds some threshold. But what metrics does it look for and why? How are boundary conditions triggered? What values of the data set off the triggers? In this section, we answer these questions about the IMCS implementation. First we talk about the types of measurements that we take. After that, we talk about how we summarize the data we collect and how determine if performance has slipped out of our boundaries for acceptable performance. We will briefly discuss the use of robust statistics in the network performance space and then discuss how all of the data collection, analysis, and out of bounds checking modules work together.

4.1.1. Types of Network Metrics

IMCS has three distinct modules it uses to measure network performance: *Timeit*, *IP Stats*, and *Imeter*. In this section, we discuss these collection agents that we use. For each module, we will talk about how we collect network performance data with it, why we use this module, and what metrics we derive with it.

Before moving on to what we measure, we should mention how the IMCS measurement systems are positioned. A system within each firewall complex does the network performance measurements and displays the results on the web. As a result, IMCS monitoring at each gateway is independent from another, and the system keeps functioning if one or more Internet gateways are unavailable.

4.1.1.1. The Timeit Module

Timeit [5] is the name of a Perl script which fetches web pages from a specified list of URLs and then provides summary information about the transfers. Timeit is also the name of the IMCS module that uses the timeit script to determine performance of web connectivity between Intel and the rest of the world. Since a vast amount of Internet use is web use, measuring HTTP Get operations gives us some idea of how the Internet functions for users of our web sites and for our internal users use of the Internet.

Many pieces of information are collected from each set of URLs fetched. For each URL, we measure the length of time it takes to perform a DNS lookup of the site we will access and the time it takes to setup a TCP connection to that site. And once we've connected, we measure the rate at which we receive the web page in bytes per second. For the times that we cannot get the web page, we record the failure rate. When we retrieve a URL, we only get the initial page and not any graphics referred to by the page. This is so we can keep the URL fetching process uniform between different web sites.

We look at each of the results for a reason. DNS lookup time can be a significant factor in web performance. We measure it in order to track whether it becomes a problem. Connect time is often association with network delays. Measuring rate as opposed to download time is a way to make comparisons between large pages and smaller pages fairer. Finally, error percentage tells us how much the HTTP gets were successful. The download rate must always be looked at together with the error rate. A high download rate might seem good but it isn't good if accompanied by a high error rate.

From the URL set results, timeit reports DNS lookup times at the 50th and 75th percentile values. This shows the central tendency of DNS lookup performance as well as slower lookup times. Timeit reports connect times in the same way, at the 50th and 75th percentiles. Transfer rates are reported using the 25th, 50th, and 75th percentile values. The number of URLs that timeit attempts to fetch and the percentage of failed requests are also reported as metrics. We have an estimated user experience metric which ranges from "poor" with a value of 1 to "good" with a value 5. This statistic is based on the connect time.

Statistics must be based on meaningful input in order to produce meaningful results. In the case of timeit, each set of URLs must be large enough so that variances in one fetch do not paint an incorrect picture of the performance whatever part of the world we are measuring. Each set needs a large number of URLs, picked from a wide variety of sources. We measure 48 sets of urls with between 1 and 78 urls in each set. The set with a single URL is a special URL to the local host. This measurement gives us some idea of the loading of the IMCS measurement system.

In addition to measuring the web performance from our customers and partners to us, we care about performance of web access from our employees to the

Internet. At Intel, web proxy servers are part of our Internet firewall strategy. While having timeit monitor web performance to many parts of the world is useful for troubleshooting employee Internet access problems, it doesn't show what performance is like through the proxy servers. Thus we have timeit do measurements through the proxy servers.

Four of the 48 sets of urls that we have timeit monitor are proxy urls. In each of our Internet gateways that have proxy servers, we monitor performance through the local proxy to 42 sites within North America. We also measure performance to the same sites without going through a proxy. This allows us to pinpoint whether proxies are behaving slowly or the Internet in general is not being very responsive. If we see a drop in rate for proxy downloads while we see no drop in direct downloads, we know that there is something wrong with the proxies.

One of the main criticisms of using HTTP Gets, like timeit does, for network monitoring is that server loading becomes a factor in the numbers you can obtain. We get around this by measuring the same sets of URLs from multiple locations in roughly the same time period. Doing this washes out the local effects of a server. If a server is slow, all of the Timeit fetches should reflect that slowness. If some are slow and others are not, the slowness should be attributable to network conditions.

4.1.1.2. IP Stats

How much traffic is passing through our firewall in a single day? Who is using it and for what? How much traffic went through in the last 15 minutes? Are we running out of capacity? The *IP stats* module helps answer these questions.

The IP stats module collects data from the IP accounting table of each firewall complex's outer firewall routers. It gathers byte and packet counts of IP source and destination address pairs. We collect this data by doing the following:

- A cron job runs every 5 minutes and runs an Expect script that does a "show ip accounting" and a "clear ip accounting." This generates a list of source and destination IP addresses with the corresponding byte and packet count between the two.

- Every 15 minutes, this data is aggregated and reported via the web interface.

- Results are also copied into a daily log file.

Because the accounting data contains source and destination information, byte, and packet counts but not port information, we must infer what type of traffic was passed. We maintain a list of specific servers and a list of special networks within the company.

By categorizing the traffic based on these lists, we can begin to understand the who and the what parts of our earlier question. Noticing whether the Intel address appeared as the source or the destination tells us whether the traffic was inbound or outbound.

The IP stats module does the following categorizations:

- **Determines whether traffic is inbound or outbound.** By examining whether the source address or the destination address was that of a host inside the firewall, we can see the direction of traffic. This indicates whether the byte count should apply to the inbound or outbound statistics.

- **Categorizes the type of traffic.** As previously stated, we only have source and destination information, not destination port number. But because we tend to keep certain types of traffic on specific hosts, we can assume that traffic caused by any given host was of a specific type. We don't know exactly which packets were passed with a destination port of 80, but because the IP address was that of a web proxy server, it was probably due to web traffic. Similarly, we can count and categorize traffic as mail, DNS, USENET News, or something else.

- **Categorizes the site.** In addition to the type of traffic, we want to know which sites within Intel are generating traffic and if so, how much. Changes in routing can cause one site's traffic to flow in one Internet gateway and out another. By monitoring for this condition, IMCS can warn us of broken routing conditions. If we see lots of Santa Clara traffic entering through our Chandler Arizona gateway, there is probably something wrong.

Once the daily logs from each IMCS system are collected and aggregated, further calculations can be made. This information is useful when determining which facility needs additional Internet bandwidth, or perhaps which facility needs its own Internet connection.

In order to make all these assumptions, we must maintain a list of which networks within the company are in use at which facility, and which servers are used for what purposes. Sometimes, instead of categorizing based on a specific host address, we categorize based on the subnet address. For example, the people running our www.intel.com servers install and replace their servers at a rapid pace. Those servers have their own subnet, so any traffic due to servers on that entire network are thrown into the www.intel.com bucket. This keeps us from having to update our server list at the rate that they make changes.

This brings us to the topic of a very important traffic category, the amount of traffic that can be attributed to www.intel.com. There is a steady stream of traffic through our firewall due to people on the Internet visiting our www.intel.com servers. It is a corporate objective that the quality of this web site be maintained at a very high level. If the quantity of traffic due to the web servers falls below a threshold, it might mean that there is something wrong with out Internet connection. Outages for the www.intel.com web site are high visibility problems, and quick action is required. In addition, we use these statistics to determine when we need to upgrade the circuits which connect www.intel.com to the Internet.

4.1.1.3. Imeter

Imeter (first known as Lachesis [7]) is intended as a general measure of the quality of Internet connectivity. It works by sending out ICMP echo-requests to key Internet landmark sites and measuring delay and packet loss (ping). We include in our list of key Internet landmarks sites that Intel employees access a lot as well as key parts of the Internet infrastructure such as the root name servers. Clearly Internet application performance will suffer without good access to the root name servers.

The metrics that we derive from Imeter are *percentage of loss packets* and *median packet delay*. Like HTTP download rate and error rate in the Timeit Module, these metrics are complementary and must be studied together. Low packet delay must not come at the price of high loss. One solution to loss in a network is to make packet queues longer. This results in more delay.

4.1.2. Data Summarization and Boundary Analysis Algorithms

Now that we know how IMCS measures different aspects of network performance, what do we do with the data that we generate? There can be a tremendous amount of raw data associated with each IMCS metric. A single metric might have many data points taken in a single collection interval. To handle all of this data, IMCS aggregates raw data into summary statistics. We often use summary statistics of metrics, making use of the seventy-fifth percentile, median (50^{th} percentile), or fractional representation of collected data. These statistics, known as Response Summary Statistics (RSS), are then run through an Out of Bounds (OOB) checker to determine whether a boundary condition has been violated.

The OOB checker reads a configuration file to apply one or more bounds checking algorithms on an RSS. Currently, there are three algorithms: *Static Value*, *Delta Fraction*, and *Delta Shift*. Each algorithm compares the current RSS value with the previous RSS value, using a *shift* value, *A/B* value, and *last_result* flag to determine if the current RSS value is "out of bounds" with respect to historical data. We will explain all of how these variables interact with each algorithm.

The *shift value*, in the context of the Static Value algorithm, indicates the maximum or minimum value that an RSS can have without triggering an out of bounds condition. With the Delta Fraction algorithm, the shift value specifies how much of a fractional difference can occur between two consecutive data points without triggering. Another way to think of this algorithm is that the shift value is the percentage that two data points can differ by without triggering an alert. With the delta shift algorithm, the shift value is maximum amount that two consecutive RSS data points can differ by without trigger an out of bounds condition. A shift value indicates how much an RSS is allowed to change relative to the previous data point. The shift boundaries are absolute limits as with the Static Value algorithm. The shift value is a fixed number with the Delta Shift algorithm. It is a fraction with the Delta Fraction algorithm.

The *A/B* value indicates if the shift the specified shift value applies towards downward or upward trends in the data. Finally, a last_result flag is initially cleared, and set when an out of bounds condition occurs. It is used by the OOB checker to ensure that an alert doesn't happen unless the trend happens over more than two samples, avoiding frivolous alerting on one-time exception data.

We create the different RSS boundary checking algorithms to deal with a variety of situations. If we felt that a particular RSS metric was well controlled and understood, then we would set up hard boundaries with the Static Value algorithm. A state table for the Static Value algorithm is shown in Table 1.

Table 1: State table for the Static Value algorithm

If A/b is	And RSS is	And last_res is	Action is
A	> shift value	0	last_res = 1
A	< shift value	1	last_res =0
A	> shift value	1	Send alert
A	< shift value	0	No action
B	< shift value	0	Set last_res=1
B	> shift value	1	Set last_res=0
B	< shift value	1	Send alert
B	> shift value	0	No action

If an RSS metric was not as well understood, we would use the Delta Shift algorithm. Table 2 contains the state table for the Delta Shift Algorithm.

Table 2: State table for the Delta Shift algorithm

A/B	If RSS(t) is	Last_dat +	Last_res is	Action is
A	<	Shift value	0	Last_dat= RSS(t)
A	<	Shift value	1	last_dat = RSS(t) last_res = 0
A	>	Shift value	0	Last_res= 1
A	>	Shift value	1	Send alert
B	>	-Shift value	0	Last_dat= RSS(t)
B	>	-Shift value	1	Last_dat= RSS(t) Last res=0
B	<	-Shift value	0	Last_res= 1
B	<	-Shift value	1	Send alert

If we did not have a good understanding of the behavior of the metric, then we would tolerate much wider swings by using the Delta Fraction algorithm. Tables 3, contains the state table for this algorithms.

Table 3: State table for the Delta Fraction algorithm

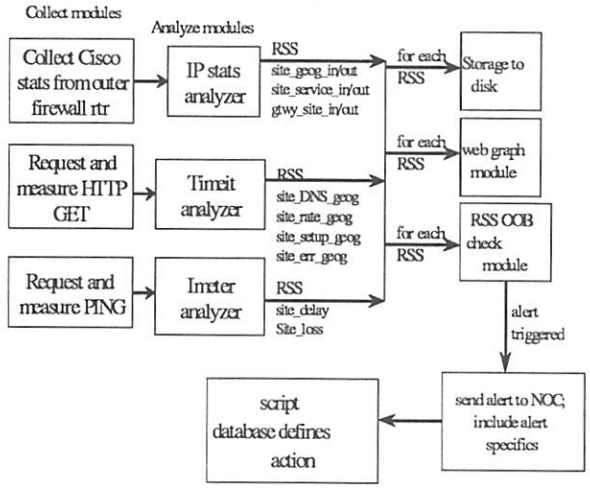
A / B	Multiply RSS by	If column 2 product is	Last_dat multiply by:	And last_res is	Action is
A	(1 –shift value)	<	(1 +shift value)	0	Last_dat= RSS(t)
A	(1 –shift value)	>	(1 +shift value)	1	Last_dat= RSS(t) Last_res= 0
A	(1 –shift value)	>	(1 +shift value)	0	Last_res= 1
A	(1 –shift value)	>	(1 +shift value)	1	Send alert
B	(1 +shift value)	>	(1 – shift value)	0	Last_dat= RSS(t)
B	(1 +shift value)	>	(1 – shift value)	1	Last_dat= RSS(t) Last_res= 0
B	(1 +shift value)	<	(1 – shift value)	0	Last_res= 1
B	(1 +shift value)	<	(1 – shift value)	1	Send alert

4.1.3. Robust Statistical Treatment of Data

You may have noticed that we have been using percentile statistics and have not used terms like *average* and *standard deviation*. It has been well established in the network measurement literature [2,3,4] that Gaussian (normal) or Poisson distributions do not represent network performance data well enough for planning or predicting purposes. Network data are characterized as "self-similar" [2]. Willinger's paper clearly demon-

strates that the Central Limit Theorem does not apply to network traffic data. As such, the traditional process control summary statistics of mean (for central tendency) and standard deviation (for variability) are not valid when applied to network data. Thus, for the IMCS the central tendency is summarized by the median or 50th% of the aggregated data. The spread between the 75th and 25th% of the aggregated data quantifies the variability. The spread between the 75th and 25th% is called the inter-quartile range (IQR). The 25th, 50th and 75th percentiles are plotted on one graph to minimize the number of graphs that need to be presented and to enhance understanding the customer experience. Traditional statistical process control methodology uses mean and standard deviation or X-bar and R charts for graphically presenting summary results of central tendency and variability. As most network engineering staff are less familiar with quantifying variability, IMCS plots the 25th and 75th percentiles instead of the IQR.

Figure 1: IMCS Data Flow



4.1.4. Data Flow between Modules

Earlier, we talked about how Imeter, Timeit, and IP Stats generated data with separate data collection modules. We also mentioned how data was fed into the OOB checker to determine out of bounds conditions. How does all the data flow between IMCS modules. We show the data flow and key metrics in Figure 1. The raw data generated from the collection modules are transferred to analyzer modules, which compute data values RSS. The RSS data values are stored to disk, sent to the graphing module for display, and piped into the OOB checker for alerting. If an out of bounds condition is detected, then an alert is sent to the Intel Network Operations Center (NOC).

4.2. Monitoring IMCS Metrics and Responding to IMCS Alerts

Figure 1 shows that when an out of bounds condition is detected, an alert is sent to Intel's NOC. What happens when that alert is sent? How does the NOC know what to do? This section describes how we implemented the user interface to IMCS and how alerts are processed.

Once IMCS generates an alert, an alphanumeric page is sent to the Intel Network Operations Center describing the RSS that is out of bounds. If you look at figure 1, you will notice that each RSS metric contains the site name and the type of metric (loss, delay, HTTP rate, etc.). NOC personnel execute a script based on the type of metric. For example, if a delay alert is generated, there is a specific script that must be done that is different from say, an HTTP rate alert. The site information lets the NOC know where the problem lies and who to dispatch to fix the problem.

The main interface to IMCS is a web page that we call "The Big Picture." The Big Picture is a snapshot of all of Intel's Internet connectivity in one screen. This web page is a large table where each row is a different type of metric (e.g. packet delay) and each column represent a firewall complex. In each cell of the table is a thumbnail graph of the appropriate metric for each firewall are shown. The thumbnail shows a rolling 24 hour graph for that metric. This allows someone looking at the big picture page to see a comparative view of the metric from each firewall. At the start of on each row of the Big Picture is a script that describes how to handle alerts for the metric.

Across the top of each column is general information about that particular firewall complex. There is a web link to the history of alerts for each gateway, as well to as IMCS System Status for each gateway. If a gateway IMCS system is having a problem then its System Status turns red. Otherwise it is green.

Each miniature "thumbnail" graph on the Big Picture is linked to a web page containing more information about that gateway's metric. This makes it easy to quickly drill down and get more information about a problem or look in detail at a metric. On these more detailed pages, we display graphs of rolling 24 hours, the current week, the last week, and the past 20 weeks for that particular metric. We can also get the actual

data points for recent values of the metric. This is useful when you need real data values for troubleshooting a problem or for tuning the alert threshold values.

IMCS scripts usually have the NOC personnel check if an out of bounds condition is local or Internet wide and change their response accordingly. On each metric row, there is a pointer to the proper script that is to be executed when an alert for that metric is received.

The modular construction of IMCS and its geographically distributed monitoring systems lends itself well to using the web as the primary reporting and access mechanism. In order to remain somewhat browser and HTTP server independent, all of the IMCS display and query functions are built on standard HTML pages, with frames and JavaScript used sparingly to provide easy control, and common CGI Perl programs to remotely generate HTML pages based on GET-mode queries. Debug programs such as ping, traceroute, and whois are implemented as CGI programs. The use of GET-mode CGI queries allows you to copy an URL to a mail message, send it, and have someone else see the results of the query too. It also enables us to embed actual pings or traceroutes into a script HTML page. We can have a link such as "ping the gateway interface" point to a URL that actually pings the gateway interface and return the results.

While each IMCS system creates HTML pages and graphs/thumbnails for each metric on its own document tree, a special setup was required to produce the "Big Picture" that would accommodate the potential outages of a remote IMCS system, and display the overall health of the firewalls. The page containing the Big Picture is not on several servers – only one. We discovered that browsers don't always flush graphics from the browser cache (even with a Pragma: no-cache, and all the other standard tricks), unless the web-server also emits an Expires: header for the graphic. Our previous web servers did not have an Expires option so we use the Apache server, which does have the Expires option. A cron job was constructed to run every three minutes to grab the latest thumbnails for a Big Picture display. This job refreshes the graphics for display and reports possible problems with the measurement process.

4.3. Other implementation highlights

IMCS is implemented with PERL, C, and Expect. The analyzers, OOB checker, pager, and graph generator are done in PERL. The PERL graphing code makes use of *GNUplot* and *pbmtools* to generate GIF files. Packet delay and loss data are collected by *fping*, a C program. Router statistics are collected by an Expect script.

Individual modules are configured with text files on the file system of the IMCS monitor. Only one configuration file exists for a particular module – it is edited and stored in one central location and pushed to IMCS machines through our Make driven update process [8]. Processes on each machine go through a rule set to see which configurations and directives apply to the current process.

As we mentioned above, detail pages are created and updated for every metric that IMCS measures. This includes metrics displayed on the "Big Picture" as well as many more metrics which are not on the "Big Picture". Most of the additional metrics target specific countries or regions of the world. These metrics are used to determine the network performance of our electronic commerce activity and allows us to monitor and improve that performance.

We mentioned above that recent data is available in text format via a "View recent data" web link at the top of each detail page. Incidentally, the simple text format is how IMCS stores its data, and is what the graph_rss tool reads to generate its graphs. Each data point is represented by a line of text with two whitespace separated fields. The first field is a UNIX time value, and the second is the value of the metric at that point in time. New data in this format is easily appended, and graph_rss uses a binary search to quickly find the start of data it needs.

In addition to graphical and textual view of data, we have a web tool that lets us do side-by-side comparisons of full sized graphs. We also have a "My Big Picture" tool to construct custom display of thumbnail graphs. These tools are extremely useful when we need to compare IMCS metrics not on the "Big Picture".

5. Experiences with IMCS

What are Intel's experiences with IMCS? Our experiences have been generally very good. We have found that IMCS is effective in finding problems and extremely extendable. We did encounter a number of other issues, though. In this section, we will detail our experiences with IMCS, starting with ability to find problems, describing its extensibility, and talking about other issues that we ran into.

5.1 Discovering Problems

IMCS is pretty good at finding problems. Imeter delay and loss usually were the first indicators of line down or congestion conditions. This is because Imeter's role in monitoring general conditions and its relatively frequent 10 minute polling interval. Figure 2 shows periods of high delay. In this particular case, we noticed that there we began to experience high periods of delay after one maintenance window.

Figure 2: Delay problems after network "maintenance"

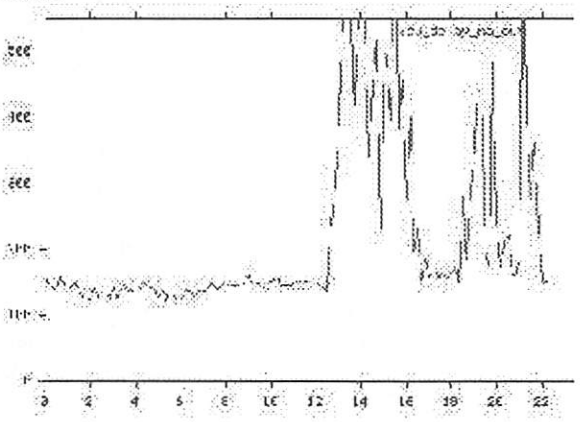


Figure 3: Proxy HTTP Error Rate During Network Problem

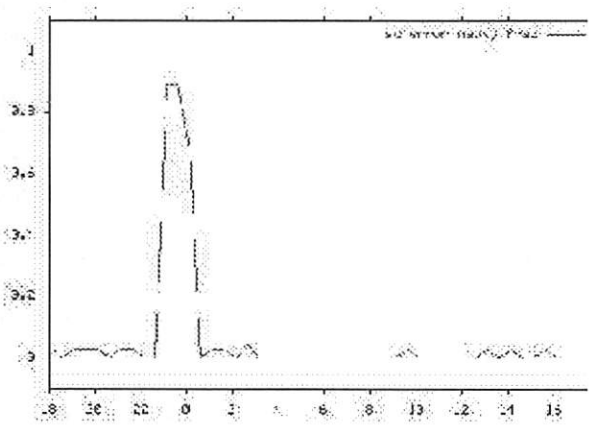


Figure 3 shows a high error rate for proxy HTTP get requests. This particular incident occurred during a problem with a DMZ network segment. The graph shows the start and end of the problems as errors increase during the episode and decrease after the segment was fixed.

As previously noted, timeit is run against URL sets that represent geographic diversity within a country and industries relevant to Intel business foci within the same country. Prior to deploying significant business applications in a country, IMCS results are evaluated against criteria known to result in relatively good performance. If the country's performance is expected to result in poorer performance for Intel's business application customers then effort is applied to find and fix possible sources of problems. Such an effort was undertaken with the main regional ISPs for the country shown in Figs 4 and 5.

Figure 4: Performance to City 1 Before and After Corrective Effort

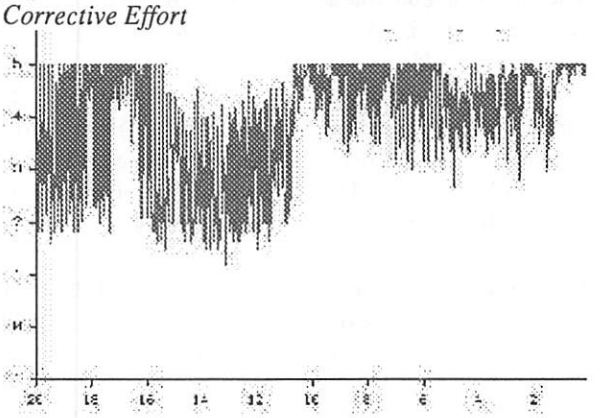
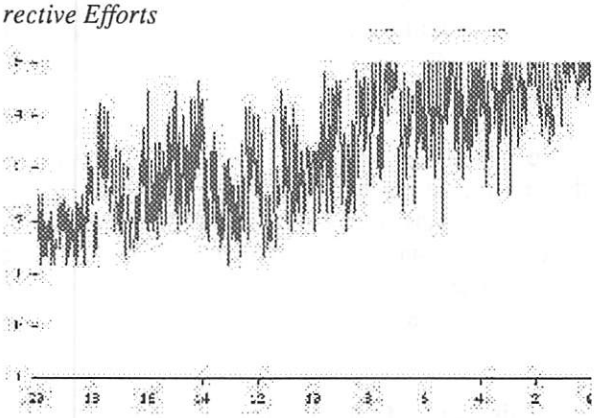


Figure 5: Performance to City2 before and after Corrective Efforts



By early December 1998, one region showed a substantial improvement in the time of day variability of performance as measured by the narrowing of the daily highs and lows in each of the 25th/50th/75th percentiles for the transfer rate and the decrease in connect times. The other region for the country experienced gradual

improvement over a four week period primarily in the connect times. Since all three of the 25th/50th/75th percentiles were impacted at the same time it is very unlikely that all 18-20 target sites improved simultaneously. This suggests network infrastructure improvement. The Intel timeit measurement tool measured other countries at the same time. No other countries showed improvements simultaneously with this country's improvements. This suggests the changes were not due to Intel measurement or network systems. A request has been made to identify the infrastructure changes implemented by the country's ISPs but as of this writing they are unknown.

5.2 Extensibility

The modular construction of IMCS makes it easy to add a new metric to the system. A recent example of this was the addition of a User Experience metric (*usrexper*) which is based on a categorization of the timeit connect metric to yield a number from 1 to 5 indicating the probable user's experience (1=poor, 5=execellent). To add the metric to timeit processing to the IMCS systems we did the following:

- Modified the timeit_rss program to compute the new usrexper metric <gateway>_usrexper_<region>_estim
- Modified the graph_rss program to handle graphing of the usrexper metric.
- Modified the timeit_pretty program to generate the HTML index page for timeit to include the new usrexper pages.

Since graph_rss generates both the full-size and thumbnail size graphs of each metric, and the HTML pages, the results were available at once. To fill in the historical data for usrexper, the <gateway>_connect_<region>_[50|75|iqr] files were processed (one time) to yield the usrexper streaming files on each IMCS system.

By adding a new CGI script on the IMCS system, you can make available new debugging tools. The best example of this was the addition of the Looking Glass functionality (based on the Perl scripts from Digex). This script allows a query to the router to be performed without requiring the use of the router's access password(s) by the end-client. The passwords are provided by two scripts (aetna and lemnos) which are not executable directly via CGI, but are invoked by the CGI programs as part of the Looking Glass functions.

A modification to the Looking Glass program allowed us to write a router-based Imeter (where delay and stats are gathered from PING on the ISP router). The results were processed by the Imeter RSS code to create new streaming files and graphics. Router based Imeter allows us to compare performance between ISPs at the same gateway.

One of the more interesting extensions of IMCS is letting IMCS systems monitor their own performance statistics. Our IMCS boxes currently track their own load averages and CPU idle time and make that data available on the web.

When the 'My Bigpicture' function was created, a new framed HTML page set was created, with JavaScript enabled selection functions to invoke the new my_bigpicture.pl CGI script. The support CGI script (available_metrics) was deployed to the IMCS systems, and a get_available_metrics_list.pl program is run (when the metrics are added/deleted) to create a control file (available_metrics.ini) for use by the my_bigpicture.pl program.

5.3 Issues Encountered

We ran into a number of problems with IMCS. The first problem is really part of any kind of alert system based on limits. It is difficult to pick limits that are effective. The only way that you can achieve good boundary conditions is to guess at some limits, and see if you get false positive alarms or see false negatives situations. If you get false positives, you need to make your limits less stringent. We wanted to avoid too many false alarms because this results in the NOC personnel no longer taking IMCS seriously. If we see false negatives, then we need to make the boundary limits more stringent. False negatives mean that the tool is not doing its job.

As conditions change, the metric limits need to change too. The most common condition we encountered with this is with Imeter delay limits. As Internet usage increases at a site, the site's line to the Internet becomes increasingly congested. We see this as time of day swings in delay. Because there is usually lead time in getting bandwidth upgrades, we have had to change the metric limits to reduce the number of alerts generated. Congestion caused alerts are not useful after we have learned that the line is congested and needs upgrading. After the line upgrade and conges-

tion goes away, we go in and make the delay limits more stringent.

The way we implemented the trigger limits continues to cause us problems. The trigger limits for a metric and the last datapoint for the metric are stored in the same file. This makes it impossible to centrally manage and distribute trigger configuration files because the contents of the file containing the limits is constantly changing.

A final problem we encountered is with active measurements. To measure performance, we generate live network traffic. Some sites turn off ICMP access in fear of ICMP based attacks like SMURF [9]. This reduces the number of landmarks that an ICMP echo based tool can monitor. ISPs also have the ability to affect the priority of ICMP packets in their networks. Given that the typical action would be to reduce the priority of ICMP packets, Imeter would be less reflective of actual user traffic in this such networks.

To eliminate the effect of web server performance, we measure performance to identical web sites at roughly the same time from all of our Internet gateways. As a result, each measured web site gets multiple hits from Intel measurement system. We have received a few complaints from web sites that we are measuring. IMCS measurement traffic can be an issue if the web sites have fees based on the volume of traffic they generate. In such cases, we usually remove the web sites from our URL lists.

6. Future Improvements

We are planning a number of additions and improvements to IMCS. These include monitoring additional services, finding new ways to measure performance, and modifying its alerting functionality.

Other Internet services are promising targets for IMCS monitoring. In particularly, we are looking at focusing IMCS on SMTP relaying and DNS services. For SMTP, we want to monitor queue depth, propagation delay through a server, message traffic, and message size. Coupled with monitoring system performance, looking at these will give us a thorough view of how well our SMTP relays are functioning. For DNS, we want to monitor response time for response to general queries and for queries of the zones that our DNS servers serve.

Since active measurements have a number of drawbacks, we want to start focusing more closely on passive measurements. SMTP relay logs, web server logs, and proxy server logs contain a wealth of information that can be mined for performance information without generating live network traffic. We also want to start looking at other forms of passive data such as netflow export data from cisco routers.

Finally, we want to improve the way that we do alerts. Paging has its problems as we have mentioned, and we want to find other ways of notifying NOC staff and others, such as screen color changes and e-mail.

References

[1] "Intel Sets Commerce Record." *InternetWorld.* November 23, 1998.

[2] Leland, Will E., Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson, "On the Self-Similar Nature of Ethernet Traffic," *Proceedings ACM SIGCOMM 93*, 13-17 September 1993

[3] V. Paxson, "Empirically-Derived Analytic Models of Wide-Area TCP Connections," *IEEE/ACM Transactions on Networking*, 2(4), pp. 316-336, August 1994.

[4] Paxson, V., G. Almes, J. Mahdavi, M. Mathis. "Framework for IP Performance Metrics." RFC 2330, May 1998

[5] Lidl, K, ftp://ftp.va.pubnix.com/pub/uunet/timeit-2.1.tar.gz, October, 1996.

[6] Libes, Don. *Exploring Expect.* O'Reilly and Associates, Inc., Sebastopol, CA 1995.

[7] Sedayao, J. and K. Akita. "LACHESIS: A Tool for benchmarking Internet Service Providers," *Proceedings of the Ninth System Administration Conference (LISA IX)*, Monterey, California 1995.

[8] Hambridge, S., C. Smothers, T. Oace, J. Sedayao. "Just Type Make! - Managing Internet Firewalls Using Make and other Publicly Available Utilities." *Proceedings of 1st USENIX conference on Network Administration.* Santa Clara, California, April 1999. 7] Sedayao, J. and K. Akita. "LACHESIS: A Tool for benchmarking Internet Service Providers," *Proceedings of the Ninth System Administration Conference (LISA IX)*, Monterey, California 1995.

[9] CERT Advisory CA-98.01. "'smurf ' IP denial of service attack". Originally posted January 5, 1998. URL: http://www.cert.org/advisories/CA-98.01.smurf.html.

Supporting H.323 Video and Voice in an Enterprise Network

Randal Abler, Gail Wells
Communications Systems Center
School of Electrical and Computer Engineering
Georgia Institute of Technology
Randal.Abler@csc.gatech.edu
Gail.Wells@csc.gatech.edu

Abstract

H.323 is a relatively new standard for video and voice transmission that specifies using IP packets as the transport. This opens the possibility of adding an inexpensive camera to a modern desktop, and allowing two way video-conferencing between any offices so equipped. While the H.323 specification addresses LAN networks, can H.323 be used in a WAN environment? What characteristics are necessary to support H.323 in the LAN and WAN networks, and what is the impact of H.323 on other traffic in the network? This paper attempts to outline the impact of running H.323 on a network and lay out some guidelines that should be useful for accommodating H.323 on both local and wide area networks.

Motivation

This paper describes results from ongoing work to examine and potentially deploy H.323 in Georgia, done in conjunction with the State of Georgia Department of Administrative Services (DOAS), and with the support of the National Science Foundation under grant NCR-9613986. Currently Georgia operates an independent H.320 video network known as GSAMS, based on dedicated leased lines. This network currently contains about 350 nodes throughout the State of Georgia. H.320 is a commonly deployed video-conferencing standard based on switched circuit such as ISDN connections, or dedicated leased lines such as T1 circuits. Unless technology such as ATM is deployed, these connections cannot simultaneously support any other type of use while a H.320 video call is in session. Therefore expensive T1 circuits are dedicated to occasional video calls. Since H.323 uses IP networking as its transport, it can co-exist with other uses of the telecommunications circuits.

DOAS also operates a data network that supports IP, IPX and SNA for sites across the State of Georgia, with IP connectivity to the Internet. Multiple state agencies use this network for a wide variety of tasks. Therefore a tentative infrastructure for supporting H.323 already exists. It is inevitable that some H.323 use will occur across this network unless explicit steps are taken to restrict it.

Rather than taking a restrictive approach, we are investigating the feasibility of deploying H.323 over data network, and may migrate some of the H.320 sites to H.323. A dedicated test setup involving two locations, interconnected with commercial Frame Relay and a private ATM network, is being used to both technically evaluate the options and to provides hands on experience for network designers and operators, as well as demonstration capability for interested customers.

H.323 Overview

H.323 [H.323] is an ITU standard that provides for the transport of real-time voice and video over IP networks. H.323 specifies H.261, and optionally H.263 as the video compression technique, similar to H.320. H.323 can be used for voice only transport as in *Voice-Over-IP*, or *Internet Telephony*. (This does not mean that all Internet Telephony applications are H.323 compliant.) H.323 defines clients, Gateways, and Gatekeepers. A client is any device capable of sending H.323 video or audio. T.120 extensions allow data sharing over an H.323 connection. The application sharing in Microsoft's NetMeeting, which allows a remote user to see the contents of any application window, is a good example of how T.120 can be used. File transfers can also be implemented.

A Gateway is a device that translates H.323 into some other voice/video transport. Two common types of gateways are H.323<->Analog Telephone line (POTS), and H.323<->H.320. H.320 is an older standard for video conferencing. It is often implemented over ISDN, or sometimes over leased lines. In using a gateway for an outgoing call, the outgoing call is placed to the Gateway's IP address, and additionally a phone number (for POTS or ISDN) is passed to the gateway. The Gateway attempts to call the designated phone number, and connects audio, and video if appropriate. In an incoming call, the gateway can either be hardwired to contact a specific IP address, or provide

some mechanism of selecting an address. In an incoming POTS call, for example, a secondary dial tone may be used which acts like an "extension", or a PBX style interface into the public telephone network may allow direct mapping of phone numbers to H.323 client IP addresses.

Gatekeepers allow network administrators to place limits on H.323 clients. This can be used to limit the maximum bandwidth a client can use, or completely disable clients. H.323 clients broadcast for a local Gatekeeper, and if none is found, enable themselves. If a gatekeeper is found, the client will check with the gatekeeper for appropriate restrictions. Gatekeepers are appropriate for networks which need the control, but the administrative burden may be prohibitive in a small or sparsely used H.323 network.

Normal H.323 clients are point-to-point devices, i.e., you call one person. To have a conference with multiple people, an multi-point conferencing unit (MCU) is necessary. The MCU allows multiple people to participate. MCU's may operate in different modes. In a voice-activated mode, the image of whomever is talking is transmitted to the other participants. In "continuos presence" mode, participants receive a split screen image, often split into quarters, allowing four remote sites to be displayed.

MCU's, gatekeepers, and gateways implementations may be combined into one product. MCU's and gatekeepers can be implemented as software only additions to a computer. For production use a dedicated computer, typically running Microsoft NT, is preferred. Gateways can be implemented as dedicated devices or as add in card(s) in a computer, PBX, or router.

While H.323 is targeted at video-conferencing, there are other video over IP techniques. H.323 is essentially a point to point protocol, much as the existing telephone network. If your goal is to broadcast a meeting, or to have some form of video library available on demand over an IP network, then other solutions are better suited. A widely used product suite to address these needs in the Real-Networks products. These products use a proprietary encoding. Several other products exist which offer similar capabilities. If broadcast of video content is used within your network, then it is worth looking into enabling multicast IP routing. In a non-multicast IP network, a separate video stream exists from the server to each client. Multicast IP allows multiple clients to monitor one common video packet stream, reducing the load on the server and saving network bandwidth.

Test Lab configuration

Figure 1 shows the connectivity of the evaluation and testing facilities. Both sites are located in the Atlanta area, separated by approximately 4 miles. One site is located on the Georgia Tech campus and the other site is located in a state office building in the downtown area. The sites are interconnected with a T1 Frame Relay link, and with an OC3c ATM link which runs via an OC12c trunk. The T1 link is provisioned with multiple virtual circuits to allow parallel testing of multiple transports. This baseline configuration is modified as needed to add network monitors, load generators, and additional network components such as additional routers to create more IP hops.

Each site has three or more PC's dedicated to H.323 video testing. This allows multiple concurrent session to be run. Future testing will allow for the addition of Gatekeepers, Gateways, and MCUs from various vendors. The network connectivity supports testing local area network configurations using shared 10Mbps Ethernet, switched 10Mbps Ethernet, as well as shared and switched 100Mbps Ethernet. The ATM interconnectivity between sites allows testing in a configuration similar to what might be found in a high-speed backbone (campus) network, with multiple router hops and switching points. By using a circuit emulation line over the ATM network, leased line connections from 56kbps to T1 speeds are being tested. Finally, dial-in servers (not shown) allow modem-based and ISDN connections to be tested.

Typical H.323 configuration

Configuring a PC to work as an H.323 station requires both the correct hardware and software. Getting a good working combination is not as simple as might be desired. The computer audio system should support full duplex sound, which may require an upgraded sound system even in modern PCs. Echo cancellation is not good in many situations and often the best solution is to use headsets instead of speakers to eliminate the feedback. Video inputs to a desktop can use the video standard NTSC into an encoder card, or a parallel port or USB connected camera. The parallel port or USB cameras are easy to connect, but often are more limited in the image size and quality. An NTSC encoder card, plus an NTSC camera is a more expensive combination, but offers better quality, and can be used to connect to a VCR or other NTSC output device. Most of the desktop H.323 solutions rely on the system CPU to implement the audio and video compression and decompression. The CPU speed becomes a limiting factor in sending good quality video. While usable video can be achieved with a slow

CPU if a small image is sent, and a low bandwidth is specified, good performance requires the fastest of modern PC's. Slower computers will also have a more noticeable problem with audio delay. Audio delay causes two problems. The H.323 systems we tested do not delay the video and audio the same amount. This causes a lip-sync issue. Also, when the audio delay is noticeable, the dynamics of human conversation become difficult, and two people will often start speaking at once.

Initial tests were done with 200MHz Pentium systems, which could not keep up with medium quality audio and video while implementing software compression. We have found 333MHz Pentium II systems to be adequate, and achieve excellent performance with a dual 450MHz processor Dell Precision 610. The multiple processors are useful when running application sharing. Single processor machines often experience temporary frozen video, and dropped audio, when an application program uses significant CPU time, such as when an application starts. An alternative to using a faster machine is to invest in a video encoder card with hardware compression. The system CPU still implements the decompression of the incoming video, and given the cost of hardware based compression cards, $800 and up, it may be worthwhile investing in a faster computer since a faster machine accelerates other non-video applications.

Dedicated H.323 systems often include a hardware compression card with multiple inputs, allowing you to switch between multiple cameras such as a document camera, etc. Turnkey systems are appearing which support both high quality H.323 and H.320 384kbps ISDN calls.

While measuring packet loss and latency are objective measurements, video and audio quality are subjective. Video quality is a function of frame rate, consistency of the frame rate, image resolution, image noise, and compression artifacts. Noise, echo, and clarity similarly effect audio. Additionally, when audio and video are compressed, transmitted, and decompressed on the remote end, the video and audio may get out of synchronization. This is most noticeable when a person's voice does not match their lip movements, and therefore is commonly called lip-sync.

H.323 bandwidth & Packet Loss

Many factors effect the bandwidth use, such as the image size, hardware capabilities, the software product used, compression algorithm used, what network speed is configured into the software, and the image complexity and dynamic nature of the imagery. RMON probes are being used in our test configuration to provide traffic plots of the traffic generated from H.323 sites. Baseline tests show that an active desktop H.323 session can use between 20kbps for a voice only conversation to over 256kbps for a high quality desktop videoconference. Dedicated H.323 systems with hardware compression may use 2 to 3 times this bandwidth.

Most of the H.323 clients allows the bandwidth to be adjusted by the user, subject to a maximum imposed by a gatekeeper if present. Depending on the product, this bandwidth may be set in KBPS, or in terms of your connection type, such as "28.8 Modem", "ISDN", "LAN". Using the right setting is important in getting the optimal connection quality. Higher bandwidth settings allow for faster image updates. This works as long as the bandwidth is really available on the network. While it is a subjective judgement, we did not find the image quality useful for 14.4 Modem or 28.8 Modem settings. These setting are effective for audio only, and for application sharing without video.

When the H.323 configuration is set for more bandwidth than is available on the network, the resulting video and audio quality suffers drastically as the data packets are lost. This setting is often configured when the software is initially installed, and therefore gets overlooked as a parameter that may need to change on a per call basis. When calling between two sites on an uncongested LAN or Campus network, the LAN setting will give the best quality video. Using the same "LAN" setting to call a second site that is reached through a remote link such as a loaded T1 link, or an ISDN link, will result in a broken video image and audio that suffers from frequent drop-outs.

To illustrate this an H.323 call was set up between two LANs connected by an ISDN line. On the "transmitter" LAN, a high-speed desktop was connected to a VCR video source, and an RMON probe attached to that LAN. The probe output is shown in Figure 2. On the "receiver" LAN, another RMON probe was set up to measure the traffic after passing through the ISDN line. This is shown in Figure 3. While video was also sent in the reverse direction, the RMON probes were not configured to capture the reverse direction traffic. Both plots cover the same time interval, covering three separate calls. In the first call, shown on the left, the H.323 client was configured to use "28.8 modem" bandwidth. This resulted in approximately 3000 bytes per second, or 24kbps, of traffic. Audio and video quality is good, but the frame rate of approximately 1 frame every 2 seconds did not provide a sense of motion. The second call was configured for "ISDN" bandwidth. This resulted in an average of about 7500 bytes per second, or 60kbps. Video and audio quality are good, and the frame rate of approximately 3-5 frames per second provides a sense of motion and continuity. The third call was configured for LAN

connectivity. The transmitter sent approximately 30,000 bytes per second, or 240kbps. This was about twice the capacity of the ISDN line, which can be seen on the receiver plot which shows 15000 bytes per second (120kpbs). The video image was unrecognizable due to partial updates, and the audio was very garbled.

WAN Network Design Issues

One of the major issues in the WAN is that H.323 uses UDP instead of TCP. TCP is not practical to use for a video transmission protocol as TCP uses packet acknowledgements and retransmits to guarantee end to end connectivity. Due to the real-time nature of video applications, if the packet does not make it through the first time, it is more important to process the next packet then to try to delay the session while waiting for a re-transmit of the previous packets. TCP also has a congestion control mechanism that reduces the amount of bandwidth that it uses (slows down the transmission speed) when packet loss occurs. UDP does not provide this feature. Therefore if an H.323 session using UDP, and a FTP session which uses TCP are competing for bandwidth on the same link, the FTP session will slow down, but the H.323 session will continue to flood the link. This type of behavior can make H.323 (or any UDP service) a poor application to run over congested links.

Different queuing paradigms can help or hurt a networked application's performance. Modern queuing algorithms allow traffic to be manually prioritized. Of course, one of the first issues in prioritizing traffic is deciding what traffic should get priority. This can quickly get tangled up in a corporate decision making process. The final result will probably not be reasonable to implement and manage on a large network. The fair queuing techniques implemented in some routers attempts to share the bandwidth amongst IP flows. This allows an improvement in overall performance. Fair queuing is relatively easy to configure as it is generally either enabled or disabled for a given router interface. It may also provide some relief from the questions of setting policy on network bandwidth since all applications are treated equally.

Table 1and Table 2 provides a brief look at the effects of FIFO queuing vs. fair queuing router configurations on an FTP and H.323 session competing for the same bandwidth. This test was done on the same configuration as the bandwidth tests discussed earlier. When the H.323 session is configured for ISDN bandwidth, it uses about half of the ISDN links capacity. The TCP flow control mechanisms used by FTP result in the FTP session using the remaining bandwidth. In this configuration the router's queuing configuration does not matter, as shown in Table 1.

When the H.323 session is configured for LAN bandwidth, the H.323 session attempts to consume all of the available bandwidth. With FIFO queuing, the congestion control in TCP slows the FTP session down to very low bandwidth, while the incorrectly configured H.323 session uses the majority of the available link capacity. This is shown in Table 2. When the router is configured for Fair Queuing, the H.323 session and the FTP session achieve nearly the same throughput, and the FTP session is still effective.

Table 1 Queuing at H.323 ISDN setting

	H.323 @ ISDN speed	
	FIFO Queuing	Fair Queuing
H.323	7000 bytes/sec	7000 bytes/sec
Ftp	8000 bytes/sec	8000 bytes/sec

Table 2 Queuing at H.323 LAN setting

	H.323 @ LAN setting	
	FIFO Queuing	Fair Queuing
H.323	15000 bytes/sec	8000 bytes/sec
Ftp	< 100 bytes/sec	7000 bytes/sec

The fair queuing technique provides an effective mechansims to limit the impact of H.323 on other applications. It does not guarantee that sufficient bandwidth will be available to support H.323. Since fair queuing attempts to evenly divide up the available bandwidth, 19 FTP sessions and one H.323 session would each receive about 750 bytes/sec or about 6 kbps. This would result in unuseable video quality for any H.323 bandwidth setting.

LAN Network Design Issues

Given the bandwidth, packet loss, and latency issues, H.323 can be implemented in the LAN if the current network is not saturated. If the LAN is currently saturated, then H.323 will probably perform reasonably well but may seriously impact the performance of TCP based applications such as ftp, http, etc. The fair queuing techniques discussed earlier are generally not useable in a LAN environment and therefore will not help if the LAN network is running at full capacity. For saturated LAN networks, the solution to supporting H.323 will likely require adding higher speed backbones and switching architectures. This growth curve should be monitored for impact on network design issues. A network with 100Mbps trunks, and 10Mbps switched Ethernet, or 10Mbps shared amongst

a small user pool, should suffice to provide good H.323 operations. In a campus network environment, with multiple switch hops and router hops, H.323 should behave well for most uses if existing performance is reasonable. As with any application, as H.323 usage grows the network will have to be scaled accordingly or gatekeepers added to the network to preserve the network performance.

Future: Gatekeepers, RSVP, and Differentiated Services

In supporting sufficient bandwidth for H.323 on a wide area network, it may be necessary to give H.323 packets priority over other IP packets, as long as the H.323 behaves properly. Determining and assuring correct behaviour is the realm of the gatekeeper. One idea is to couple the gatekeeper to a bandwidth reservation mechanism such as RSVP or the IETF's differentiated services efforts. Since the gatekeeper is controlled by the network administrators, it is possible to police the H.323 sessions. While there are mention of RSVP enabled gatekeepers, this still requires a network which supports RSVP or differentiated services. Differentiated services is not yet a finalized standard, and RSVP has not caught on for wide-scale deployment.

Conclusions

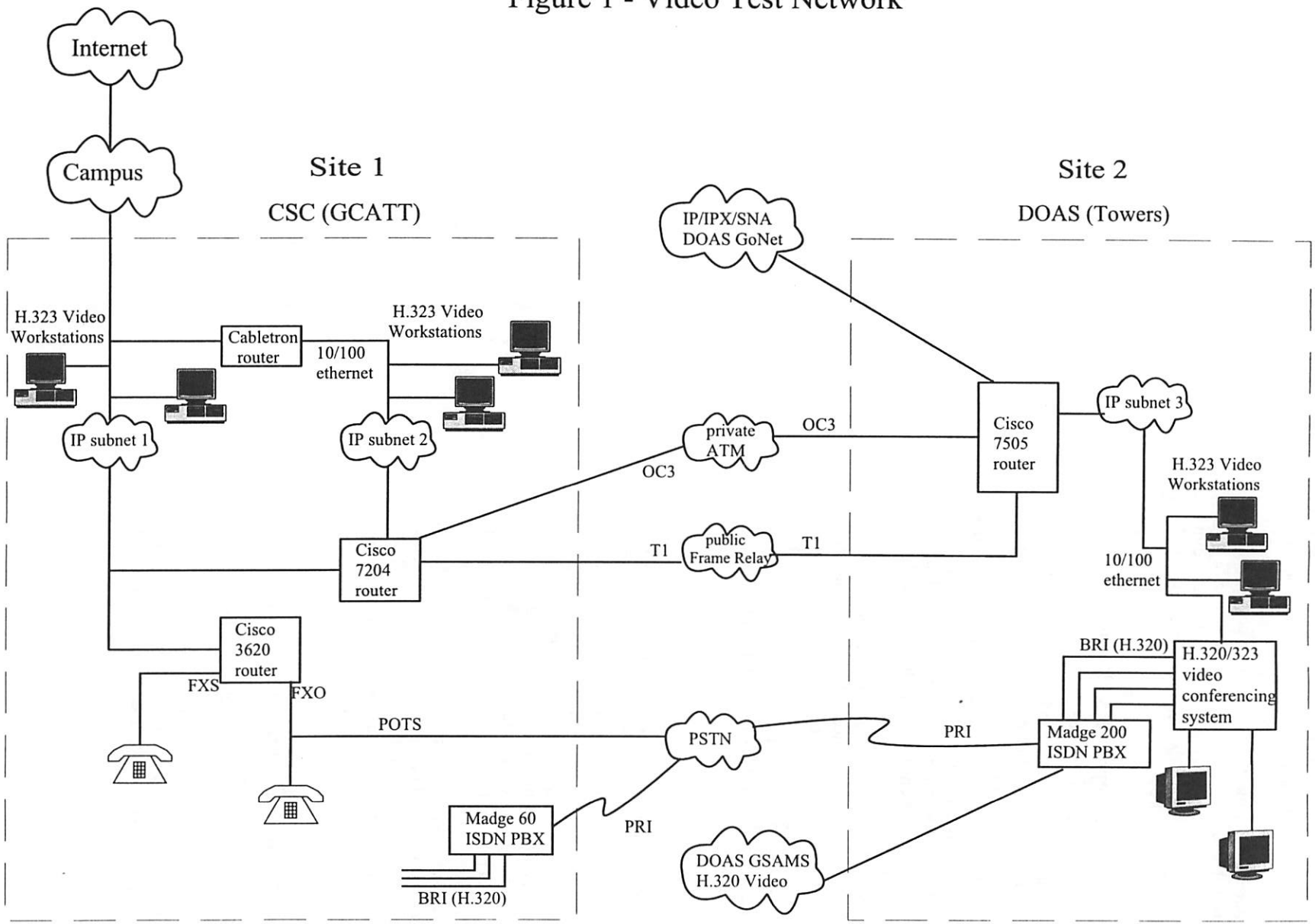
Supporting limited use H.323 in an Enterprise network is possible. Scaling it to medium use levels can be achieved with some attention to the wide-area network design. A solid network design will probably serve better than sophisticated and time consuming management techniques. Ignoring H.323 can be catastrophic since it can cause very poor performance for other network applications. Replacing the office telephone with H.323 desktops and central telephony gateways may be commonplace some day Supporting this level of reliability for H.323 over the WAN will require some form of IP quality of service, which is currently the focus of much attention in both the research world and the marketplace.

References

[H.323] Recommendation H.323, "Packet-based multimedia communications systems", 2/98, International Telcommunications Union

[DiffServ] "A Framework for Differentiated Services", <draft-ietf-diffserv-framework-01.txt>, Yoram Bernet, et al, Oct 1998, IETF Internet Draft.

Figure 1 - Video Test Network



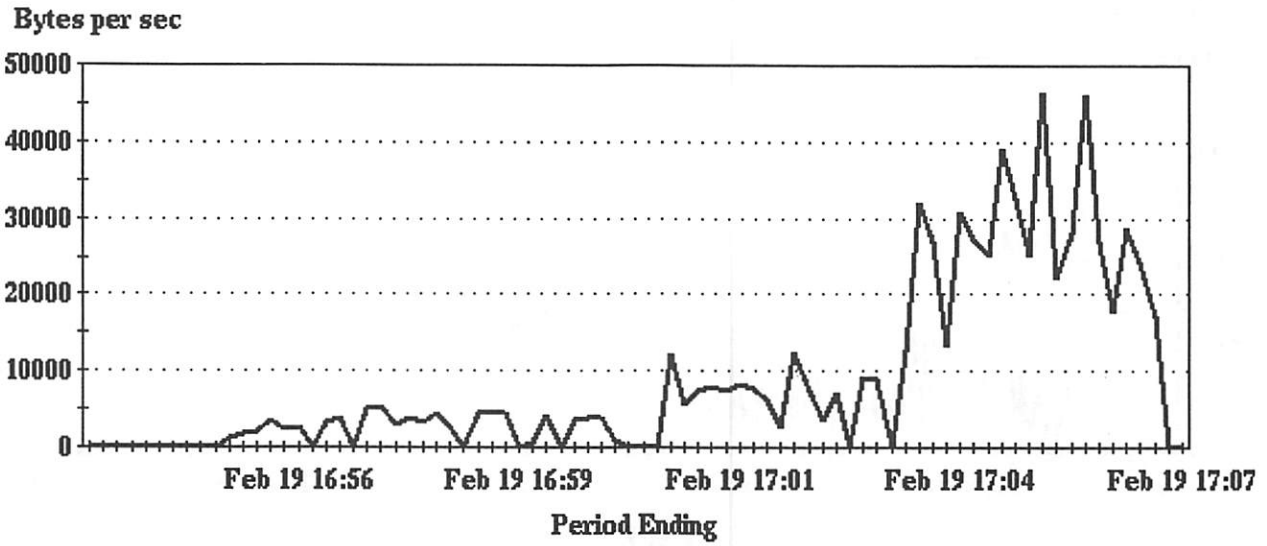


Figure 2 Transmitter Bandwidth

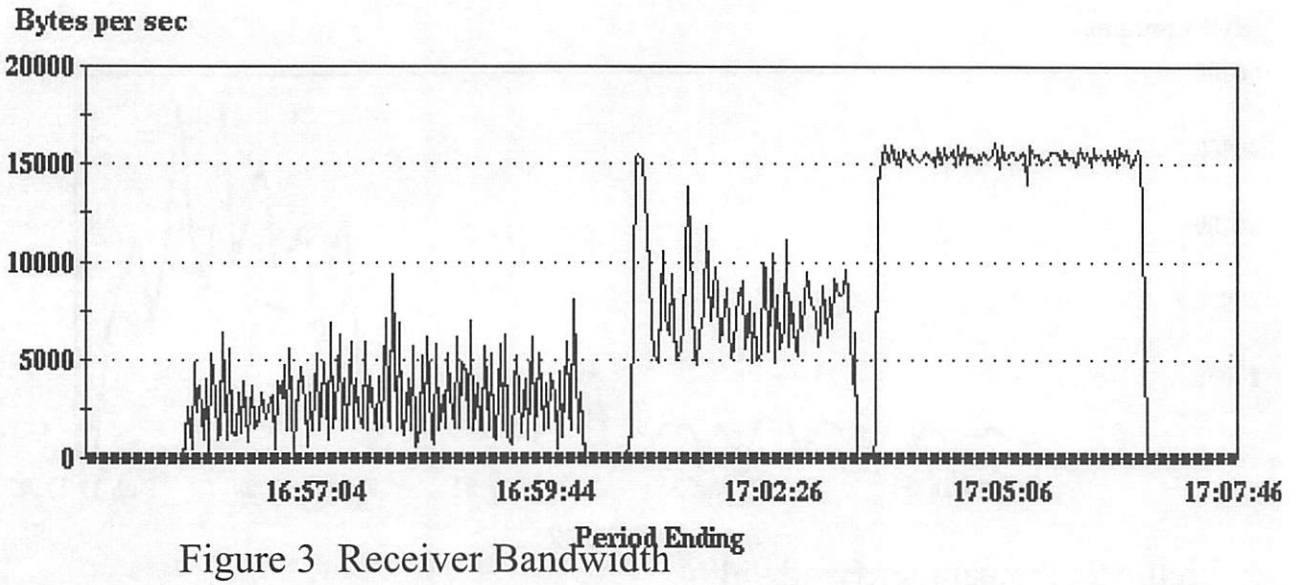


Figure 3 Receiver Bandwidth

Network Documentation: A Web-based Relational Database Approach

Wade Warner
Georgia State university
wade@cs.gsu.edu
Rajshekhar Sunderraman
Georgia State University
raj@cs.gsu.edu

Abstract

Every organization managing a network of computers has a need to organize, maintain, and access information related to the network. Users at various levels of the organization need quick and convenient access to this critical information at all times. We propose a methodology which is Web-based and which uses relational databases at the back end to store and organize this information. We envision that this Web-based system will be used in an intranet environment and will provide several levels of access to different user types.

1 Introduction

In today's complex networks, it is more important than ever that network documentation be clear, concise, and up to date. Simple tasks such as looking up the serial number of a workstation before calling tech support are often repeated endlessly because the required information is not recorded anywhere. Technicians unfamiliar with the setup of a particular workstation must first take the time to analyze the configuration of the station before any useful work can be done. Considerable time could be saved if all of the pertinent details are readily available.

Paper records are one possible solution, but there are many problems associated with them. Keeping paper records up to date is time consuming at best, and it can be difficult to search for the needed data. A better solution is to keep the documentation on the network itself.

We propose a methodology which is based on relational database technology. The information about the network is stored in a relational database (Oracle8) and is accessed via a Web browser in an intranet environment. Easy to use Web forms are used to access and update the data.

The solution presented in this paper is specific to our organization, but the system can be easily modified for any environment due to our use of Query By Example (QBE) technology for dynamic query generation. For organizations that do not have Oracle, we propose to also implement our system using a freely available database engine, mSQL.

Both, the Oracle and the mSQL implementation will be available freely to interested users.

2 Database Design

The database was first designed using the Entity-Relationship modeling technique and then converted into relational schema in SQL. The ER schema is shown in Figure 1.

The following three entity types are identified for the database:

- **Computers**: This entity type consists of all the PCs, workstations, and servers in the organiza-

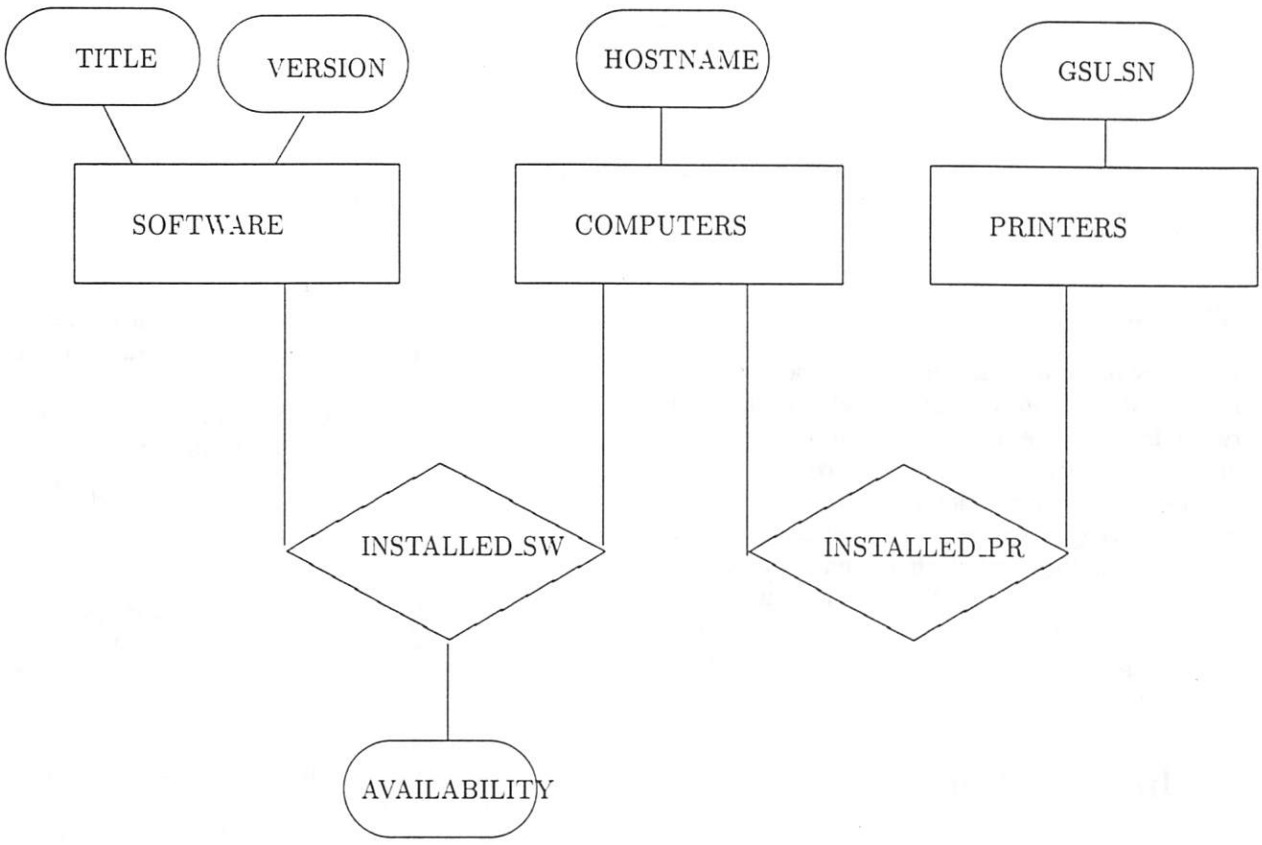


Figure 1: ER Diagram for Network Documentation Database

tion. The hostname attribute is chosen as the primary key. Three unique keys have also been identified. These are: gsu_sn the serial number assigned by the organization, ip_address, and mac_address. In addition to these attributes, several other attributes have been identified and the detailed description is shown later in the paper.

- **Software**: This entity set consists of all the software packages and programs available on the network. The title and version attributes form the primary key for this entity set. An interesting attribute identified for this entity set is the doc_url attribute which contains the URL for the documentation related to the software.
- **Printers**: This entity set consists of all the printer devices available on the network. The gsu_sn organizational serial number has been identified as the primary key for this entity set. Some of the more interesting attributes of this entity set are the model, the model number, queue_name, the queue name assigned to the printer, location, the physical location of the printer, and availability, whether the printer is available locally or on the network.

Two relationships between these entity sets have been identified. These are:

- **Installed_sw**: This is a many-to-many relationship between computers and software that indicates which software is installed (or available) on which computer. This relationship type has an attribute: availability, which indicates if the software is locally installed or installed on the network.
- **Installed_pr**: This is a many-to-many relationship between computers and printers that indicates which printer is accessible to which computer in the network.

Based on these entity and relationship types, the following SQL definition of the database tables has been developed for our organization. The relational tables that correspond to these entity and relationship sets are created in the Oracle 8 database.

```
drop table computers cascade constraints;
create table computers (
  hostname      varchar2(20) not null,
  domain        varchar2(10),
  manufacturer  varchar2(20),
  model         varchar2(20),
  architecture  varchar2(20),
  memory_size   number(5),
  hdd_size      number(7,2),
  sound_card    varchar2(20),
  video_card    varchar2(20),
  os            varchar2(20),
  vendor_sn     varchar2(50) not null,
  gsu_sn        number(6) not null,
  location      varchar2(20),
  ip_address    varchar2(15) not null,
  mac_address   varchar2(30),
  primary key (hostname),
  unique (gsu_sn),
  unique (ip_address),
  unique (mac_address)
);

drop table software cascade constraints;
create table software (
  title     varchar2(20)  not null,
  version   varchar2(10)  not null,
  category  varchar2(20),
  doc_url   varchar2(100),
  primary key (title,version)
);

drop table installed_sw cascade constraints;
create table installed_sw (
  hostname      varchar2(20) not null,
  title         varchar2(20) not null,
  version       varchar2(10) not null,
  availability  varchar2(10)
                check(availability in
                  ('local','network')),
  primary key (hostname,title,version),
  foreign key (hostname) references computers,
  foreign key (title,version) references software
);

drop table printers cascade constraints;
create table printers (
  manufacturer  varchar2(20),
  model         varchar2(20),
  queue_name    varchar2(20),
```

```
  location      varchar2(20),
  vendor_sn     varchar2(50) not null,
  gsu_sn        number(6)    not null,
  availability  varchar2(10)
                check(availability in
                  ('local','network')),
  color         varchar2(10)
                check(color in
                  ('color','greyscale')),
  type          varchar2(20),
  address       varchar2(30),
  primary key(gsu_sn)
);

drop table installed_pr cascade constraints;
create table installed_pr (
  hostname      varchar2(20) not null,
  gsu_sn        number(6)    not null,
  primary key (hostname,gsu_sn),
  foreign key (hostname) references computers,
  foreign key (gsu_sn) references printers
);
```

The next important step in the process of making this database available is to populate it with real data. For an existing network, this process may involve extracting data from existing electronic files or spreadsheets using scripts written in Perl for example, or manually entering new data.

3 Web Application Development

In this section, we sketch the Web application that will access the Oracle database defined earlier. Our Web application provides for three different levels of access. At the highest level of access, users such as the network administrator will have access to querying pages, database update pages, and report generation pages. At the next level of access, users such as managers will have access to querying and report generation pages. At the lowest level of access, casual users will have access only to limited querying pages.

Every user is assigned an userid and password to access the system. They are also assigned one of three access levels: *Privileged*, *Normal*, and *Casual*. To keep track of the user information, we have designed a table with the following schema:

```
create table users (
  name        varchar2(25) not null,
  username    varchar2(10) not null,
  password    varchar2(15) not null,
  usertype    varchar2(10) not null
              check(username in('Privileged',
                      'Normal','Casual')),
  primary key (username)
);
```

Depending on the user access level, a different set of options will be available in response to a successful login.

We now present some querying options that are available at all levels of access. Figure 2 shows three queries that the users can pose: query `computers` on host name, query `printers` on printer type, and query `software` on title.

Sample results from the query on printer types is shown in Figure 3.

4 Generalizing the Methodology

The methodology and system presented in this paper has several features that make it easy to use it in any organization.

The querying aspect of the system is divided into two categories: static or canned queries as described in the previous section and dynamic or on the fly queries.

The static queries are the ones that would apply to any organization. Typically these queries are related to getting details based on serial numbers of computers or titles of software or categories of printers etc. Almost every organization would see a need for such queries and our system provides easy interfaces to answer these queries as shown in the previous section.

In addition to such fixed queries, our system provides an interface which will allow dynamic queries to be generated and executed by the user. A visual querying interface on the Web implementing the

NETWORK ADMINISTRATION SYSTEM

GEORGIA STATE UNIVERSITY

Computer Details

Select Host Name: Hewlett Get Details

Printer Details By type

Select Printer Type: bubblejet Get Details

Software Details By title

Select Software Title: Microsoft Access Get Details

Figure 2: Query Menu

The following laser printers are available

Manufacturer	Model	Queue Name	Location	Color
Canon	C4000	cn1	745 COE	color
Xerox	X2000	xp12	744 COE	greyscale

Figure 3: Printer Query Results

Query By Example (QBE) query language is used to allow the users to pose arbitrary queries. Another Web interface for dynamic queries is based on SQL. Users who have a knowledge of SQL can easily formulate dynamic queries using this interface and have their results displayed on the Web browser. Both these dynamic querying interfaces have already been built in [6] and is easily added to the system described in this paper.

Even though the methodology presented in this paper is specific to our organization, we believe that adapting it to any other organization should be an easy task. We have tried to make the database design general enough so that many organizations can use the database tables as is. Building new functionality is then a matter of writing new code. If the database design has to be altered, some of the functions have to be redesigned and some new ones may have to be written. The dynamic querying aspects of the system is easily portable to any other organization.

The current implementation uses Oracle 8.0 database server and the Oracle Application Server 4.0 system. Any organization using Oracle can easily port our system with minimal changes. We propose to implement the system using mSQL, a publicly available popular database system. This implementation can be used by almost any organization.

5 Conclusions

We have presented a framework for documenting a local area network. A relational database (Oracle 8) is used at the back end to store the relevant information. Applications that run on the Web are developed to provide ready access to this information at various levels of security levels. Users are able to query the information using static or canned queries as well as dynamic or on the fly queries. Other users are able to print statistical information and reports from the database. Privileged users have the capability of changing the information using easy to use Web forms.

These applications are being developed using Oracle Application Server 4.0. We propose to provide an implementation using mSQL, a freely available database engine, so that organizations who do not have Oracle database server can use the mSQL system. Finally, the solution presented in this paper is easily adaptable to any organization with little effort.

6 Acknowledgments

We acknowledge the implementation effort of our graduate student, Shahnaz Ahmed.

References

[1] B.D. Brown, R.J. Niemiec, and J.C. Trezzo. Oracle Application Server Web Toolkit Reference. Osborne McGraw-Hill, 1998.

[2] R. El-Masri and S. Navathe. Fundamentals of Database Systems. Addison-Wesley Longman, 1994.

[3] B. Johnson. Oracle Web Application Server Handbook. Osborne McGraw-Hill, 1997.

[4] G. Koch and K. Loney. Oracle 8: The Complete Reference. Osborne McGraw-Hill, 1997.

[5] R. Sunderraman Oracle Programming: A Primer. Addison-Wesley Longman, 1999.

[6] R. Sunderraman ReqWeb: Relational Querying of Web Data. Unpublished manuscript, Georgia State University.

[7] L. Wall, T. Christiansen, and R.L. Schwarz. Programming Perl. O'Reilly and Associates, 1996.

Just Type Make! - Managing Internet Firewalls Using Make and other Publicly Available Utilities

Sally Hambridge
Intel Corporation
Charles Smothers
Intel Corporation
Tod Oace
Intel Corporation
Jeff Sedayao
Intel Corporation

Abstract

Managing Internet firewalls that can failover between each other is quite a challenge. When those firewalls are geographically dispersed and have a small number of people to be maintain them, it becomes even more challenging. Intel Corporation has a small staff that manages several geographically dispersed Internet firewalls with failover requirements. These firewalls use a standard screened subnet architecture [1] with packet filtering inner and outer firewall routers and a number of bastion hosts between them. These bastion hosts provide services with load balancing and disaster recovery for relaying SMTP mail, answering DNS queries, and proxying web requests. To manage this complex system of firewalls, Intel's Internet Connectivity Engineering staff have come up with a way to model all of the interrelated firewall as one distributed system. Host and router configurations are considered source to that system and compilation and installation of that source is driven by the Make [2] utility. Packet filtering Access Control Lists (ACLs) are built by a Makefile. The Makefile assembles the ACLs and executes an Expect [3] script that installs them. We configure bastion hosts by configuring Make to drive **rdist**, which run over the secure shell (SSH) [4]. In this way, only updated files are pushed out to the bastion hosts and passwords and other configuration information do not go in the clear. Our experiences with Make and these publicly available utilities are quite good - allowing us to manage a large distributed set of firewall devices. Using a Make driven approach requires much discipline, however, to avoid the distribution of bad configurations. Future plans include ACL optimization and sanity tests before and after bastion host configuration pushes.

1. Introduction

Managing a single Internet firewall complex can be difficult - there can be multiple routers with long packet filtering access control lists (ACLs) and bastion hosts performing different functions that all must work together seamlessly, efficiently, yet securely. Managing multiple Internet firewalls which interact with each other and provide failover capability between each other while maintaining both security and some comparable level of performance becomes even more challenging. The Internet Connectivity Engineering Group at Intel has created a way to manage multiple interacting Internet firewall complexes spread across the world using the familiar utility – Make utility. This paper describes how Intel integrated make with other publicly available tools to administer multiple firewall complexes across the world.

The first section of this paper talks about the firewall environment at Intel. It describes the key features of the Intel Internet firewall environment and the challenges that that led us to create our Make-based configuration tool. The second section covers how we overcame those challenges by implementing a Make driven configuration tool. The architecture of the Make tool and the publicly available utilities in our implementation are described here. The third section of the paper goes over our mostly positive experiences with our make driven update process, followed by a final section on future work that we are planning.

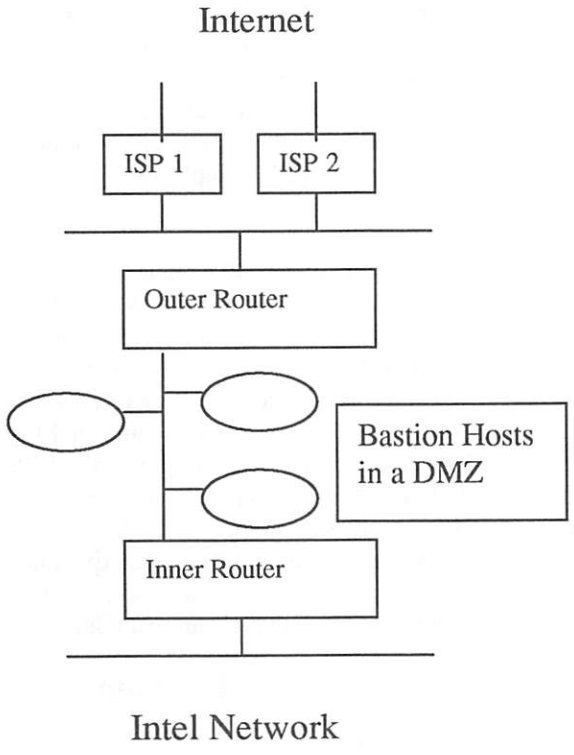
2. Intel's Internet Firewall Environment

It is impossible to understand how and why we created our Make driven update tools without understanding Intel's Internet firewall environment. In this section,

we provide context for our Make tools by describing the Internet connectivity at Intel. We also talk about the key motivating factors that led us to create these tools.

The standard Intel Internet gateway uses the screened subnet architecture [1], as shown in Figure 1. Multiple bastion hosts lie between a external outer router and an internal inner ("choke") router. These bastion hosts provide a variety of functions, such as web service to the Internet, SMTP mail relaying, proxy services (web, telnet, FTP, AOL, etc.), and performance monitoring. Outside of the external router is a segment for ISP access. Multiple ISPs have a presence (routers) on this segment and exchange traffic with the bastion hosts through the external router. Both the external router and internal "choke" router do extensive packet filtering. The packet filters prevent generic access to the bastion hosts from the Internet. The packet filters allow access to the bastion servers from hosts within Intel, but do not allow generic access from the bastion hosts into Intel. This is our implementation of "defense in depth." A single compromise of a bastion host does not mean that there is complete entry into the heart of Intel

Figure 1: Intel's Firewall Architecture



There are multiple Internet gateways at Intel, each with two or more ISPs, spread across the world. Our intention was to minimize the amount of traffic on Intel's internal network and get good performance for Internet applications. We also wanted to have multiple gateways for failover. If one Internet gateway failed, traffic should be able to flow in and out through another gateway.

Intel's firewall environment, as described above, presents a number of challenges to the staff maintaining and engineering the gateways. The first set of challenges involved the cisco routers. We had to find some way to maintain the Cisco Router packet filtering access control lists (ACLs) used on the inner and outer routers to permit failover. If one site's Internet connectivity went away, the packet filtering ACLs at other sites need to be able to allow traffic for that bastion hosts at the first sites to flow in and out. Even if there was failover, the router ACLs need to be consistent between sites. If a bastion host has a particular kind of access at its site, it should still have that access if traffic needs to fail over to another gateway. In addition, we have large numbers of cisco ACL entries. While these entries have to be consistent between gateways for failover, we have to arrange the entries to offer good performance. To make things even more challenging, if we had a problem with the ACLs we rolled out, we need some easy way to backup the changes.

The second set of challenges involves the bastion hosts. These hosts contain all kind of tables and configuration files like anti-spam lists, sendmail configuration files, and performance monitoring information. We need this information to be consistent between sites, in order to make maintenance easier and to have consistent access policies and DNS databases. It would not make any sense for one Internet gateway to have very strict anti-spam rules and another to have very loose policies. As with router ACLs, any changes in configurations need to be easy to back out, in case there are problems. Since the bastion hosts are on a segment exposed to the Internet, we are extremely concerned about problems with eavesdropping and spoofing. Any kind of updates to these hosts needs to verify the identity of originating hosts and be secure from eavesdropping and spoofing.

The final set of challenges involved the staff maintaining the Internet gateways. There are 7 Internet gateways spread around Intel but fewer people than that responsible for engineering and maintaining them. We needed an easy way to update many hosts and routers. Any personnel intensive update method would not be viable in this environment.

3. Solving our Maintenance Challenges using Make

The first step toward solving our maintenance challenges was coming up with a conceptual model for our Internet gateways. We knew that they were not stand-alone systems - instead they were interconnected and interrelated. Each gateway is an interconnected part of a single integrated system. Traffic from one gateway had to be able to travel through another. Mail servers must have consistent configurations to relay mail and block spam effectively. DNS servers must be consisent in order to provide DNS information correctly. Once we began thinking of the gateways as a single integrated system, we started looking at system and router configurations as source code with various compilation dependencies that is fed into that single yet distributed machine. With that model in mind, what better utility to manage source code compilation and installations than the classic utility Make!

Once we had a model for managing our configurations, we had to implement configuration management for the routers and for our bastion hosts. A key part was figuring out how to break down the configurations into units small enough to manipulate and build Makefile dependencies with them. The rest of this section describes how we split up configurations and used Make as a way of building configurations and installing them. The first part shows how router ACLs are managed with Make. The second part deals with managing server configurations. The final part describes how we manage our configuration files on the systems that build and install configurations.

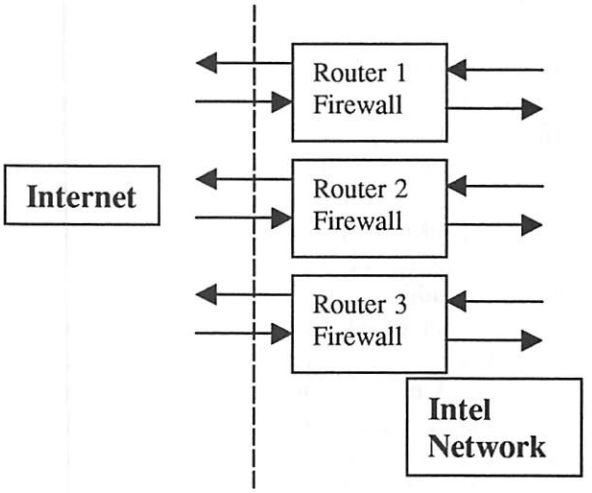
3.1 Managing Router ACLs using Make

Since a key requirement is that traffic from any gateway within Intel must be able to go into and out of any other gateway for failover, being able to manage the interchangability of router ACLs is critical. Imagine creating ACLs for a firewall complex. You have to deal with the services and well known ports for traffic coming in and for traffic going out. In addition, a gateway may have some special applications running in it to support external customers. Now imagine a second gateway with mostly the same and then merging the two sets of access lists. Doing this by hand for two firewall complexes is difficult. Doing this by hand for eight firewall complexes would be totally unmanageable.

We simplified this task by breaking down ACLs into files containing the permissions going out to each segment and coming in from each segment. By segment, we mean a subnet or network that forms a DMZ network between the inner and outer routers. The firewall architecture shown in Figure 1 shows a single segment with bastion hosts on it. In our architecture, the inner firewall router and outer firewall router each have one interface onto each segment.

We created a naming scheme based on the directionality and the router name. Thus *Intel-to-router1-seg1* contains router ACLs pertaining to traffic from Intel to segment 1 on router 1. *router1-seg1-to-Intel* contains router ACLs for segment 1 on router 1 to Intel. *router1-seg1-to-Net* contains ACLs for segment 1 on router 1 to the Internet while *Net-to-router1-seg1* contains ACLs for the Internet to segment 1 on router 1. For simplicity's sake, all files are named for the outer firewall router in each complex. As a way of illustrating how this can work, let's say that there are three geographically dispersed firewalls that need to failover between each other, as shown in Figure 2. The arrows depict traffic going in and out of each firewall to the Internet or to Intel's Internal network.

Figure 2: Three firewall complexes with failover



In this example, we have named each firewall by its outer router name. Each firewall could contain any number of segments (which are not explicitly shown here). If the Router 1 firewall should lose its connectivity to the Internet, then its traffic to and from the Internet should be able to travel through the Router 2 or Router 3 firewalls. If the Router 2 firewall lost its Internet connectivity, then Router 2 traffic should be

able to failover through the Router 1 and 3 firewalls. The same should happen for Router 3.

For each segment, we group the ACLs together that apply to traffic going in a particular direction. For example, all of the ACL entries applying to traffic from the segment into Intel are grouped together. All of the ACL entries applying to traffic from Intel to the segment are put in another group. Each group is put into a separate file. For each firewall, we can concatenate all of the files together into one large file containing all the permissions for that firewall. We also concatenate the files from any other firewall complex needing failover capability through the first gateway. For our example, assuming that each firewall complex had only one segment, we would concatenate together the following files for the outer firewall routers: Net-to-router1-seg1, Net-to-router2-seg1, Net-to-router3-seg1, router-seg1-to-Net, router2-seg1-to-Net, and router3-seg1-to-Net. The inner router at router1's firewall complex would include all of these files but also router1-seg1-to-Intel and Intel-to-router1-seg1. The inner router at router2's firewall complex would have all of the outer firewall files but also router2-seg1-to-Intel and Intel-to-router2-seg1. The inner router at router3's firewall would use a similar configuration except using router3-seg1.

Routers check packet filtering ACLs one line at a time and stop checking when they find a match. Having the lines most matched at the top speeds the traffic through ACLs and through the router. Because of this, we want the large file of concatenated ACLs to be in a different order depending on the gateway. For the router1 firewall, we'd like the files for router1 to be first in the list. For the router2 firewall, the files for router2 should be at the top of the concatenated ACL file, and similarly for the router3 firewall. We've done this by defining sets of macros for each firewall gateway which contain the files going in or going out. INFROMNET_GW1 contains Net-to-router1-seg1, and if there is any other segment on router2 (such as seg2), Net-to-router2-seg2. OUTTONET_GW1 contains router1-seg1-to-Net and if there is any other segment on router2 (such as seg2), router2-seg2-to-Net.

We then define a macro in the Makefile containing the macros of all the files for all the routers needing failover through that firewall:

```
GW1_OUTER = ${OUTTONET_GW1} \
        ${INFROMNET_GW1} \
        ${OUTTONET_GW2} \
        ${INFROMNET_GW2} \
        ${OUTTONET_GW3} \
        ${INFROMNET_GW3}
GW2_OUTER = ${OUTFROMNET_GW2}\
        ${INFROMNET_GW2} \
        ${OUTTONET_GW1} \
        ${INFROMNET_GW1} \
        ${OUTTONET_GW3} \
        ${INFROMNET_GW3}
```

If there are special permissions in GW1 which don't need failover, it is easy enough to add the file name with those permissions to the gateway macro. Here we add a special set of permissions for a segment 0 on router1:

```
GW1_OUTER = Net-to-router1-seg0 \
        ${OUTTONET_GW1} \
        ${INFROMNET_GW1} \
        ${OUTTONET_GW2} \
        ${INFROMNET{GW2} \
        ${OUTTONET_GW3} \
        ${INFROMNET_GW3}
```

If there are files to be included in every gateway, it is easy enough to define a "generic" macro which contains those files:

```
GENERIC_IN = generic-in monitor-in
GENERIC_OUT = generic-out monitor-out
```

The actual concatenation is defined by the target *router-access*. That target takes as its dependencies the gateway macro and the generic ones. It cats them together, throws them through some processing the C preprocessor and some custom filters (we will discuss this in more detail in section 4) and concludes by appending "end" at the bottom of the concatenated files.

```
Router1-access: ${GW1_OUTER} ${INTELNETS}
        cat ${GW1_OUTER} | ${CPP} \
                ${ROUTER1DEFS} | \
                intel_nets_convert ${INTELNETS} \
                > router-access
        echo 'end' >> router1-access
```

Finally, the target *router1-access-install* takes router-access as a dependency, but before it re-loads the ACLs, it also runs an Expect [3] script which pulls the current ACLs off the router and stores them with their associated counters. Cisco routers maintain a count of how many packets have matched each line of the ACL. The counters are reset each time new ACLs are loaded. In order to acquire data about the use and efficiency of our access-lists, we download the current set of ACLs before we upload the new set. Both of

these operations are done via Expect scripts which log into the router interactively, then either write out the current set of ACLs to the server and load the ACLs from the server to the router.

```
router1-acl-stats:
        /usr/local/expect/get_acl_stats router1
router1-access-install: router1-access, router1-acl-stats
        cp router1-access /tftpboot
        /usr/local/expect/load_up router1
        rm /tftpboot/router1-access
```

Recording our ACL usage has been useful for simplifying ACL maintenance. With data on which statements are used and which are not, it is easier to identify usage which is no longer needed. It's amazing that users will gladly follow a process to allow them access but develop amnesia when they need to follow a process to end that same access. Intel employees have also requested access to external applications for all of Intel and then have not publicized the availability, so these are never used as well. We remove access when the lines show no activity for three months.

The flexibility of Make allows us to use many utilities and programs as part of our process. In the example just shown, we used shell commands, expect, and PERL (within the intel-nets-convert program which we will discuss later) in order to build and compile router ACLs.

3.2 Bastion Host Management using Make

Intel's Internet gateways contain bastion host services such as DNS, SMTP mail relaying, and Internet access proxying. Maintenance of these services shares many of the same requirements as Intel's firewall routers. The services are mission critical and call for a high level of care when changes are made. In addition they also back each other up during outages and provide load balancing, thus requiring consistent configuration among hosts. Make drives our bastion host configuration too, as we will describe in this section.

One of the core ideas we utilize at Intel is called "Copy Exactly!" Once a product or facility is designed for a specific purpose, it should be deployed identically everywhere. It should not be redesigned and rebuilt every time it is deployed. Adhering to this ideal helps our chip fabrication plants come up to speed rapidly and produce very high yields. In the Information Technology (IT) space, particularly in the network and server space, it helps us create rapidly deployable services with lower maintenance. More specifically, it helps us successfully maintain 43 (and growing) geographically diverse bastion host servers with a staff of three engineers, and even leaves them time to work on lots of other things.

The key to our success is in doing things very efficiently and very carefully. We use a central secured server which holds several configuration file distribution trees. One tree contains the base operating system for all of our bastion hosts. Additional distribution trees such as *dns* and *sendmail* exist for each type of bastion host application. All distribution trees are as similar to each other as possible. The directory layouts are similar, and the commands to check and install changes are the same, thanks to a commonly included *Makefile.inc* file. Finally, access to the distribution server is granted to only those who need access. Our Makefile automatically generates what configurations are necessary, logs in to systems that need to change, and loads the configurations.

With our centralized build in place as a reference, we needed a method to keep all our servers in sync with the reference. Ease of use was very important. However, even more important, was to manage the servers securely. Building a system that could meet both of these requirements was a key challenge. We were already accustomed to managing the servers by logging in, hand editing, and setting the permissions and ownership on critical files. Keeping this familiar model would make the management task easier for us and any future administrators. We needed a tool that could clone entire directory trees between several machines. By using **rdist**, we could build and develop the master directories on the central management server. Then, we would clone the reference directory tree onto the remote servers.

This solved the ease of use problem. Now, we needed to determine how to use rdist without compromising the security of the remote server. In order for remote management to work, each remote server needed to be configured to allow our central management server to gain root access. It must do it in such a way that could not easily be exploited by anyone other than the central management server. We needed strong authentication, to avoid such things as IP spoofing. We needed strong encryption to avoid things like password snooping. Fortunately, rdist supports the use of alternatives to the *rsh* program. We found that **secure shell** (SSH) [3] worked well this purpose. It can use host-based public/private key authentication, so that only our central management system, with the right private key can gain access. It also uses

encrypted sessions to prevent someone from sniffing out a cleartext password, or connection hijacking.

Additionally, rdist allows for minor differences between servers. During the design of our system, we considered using rsync, which also supports ssh, but it lacked this last feature.

Architecturally, our bastion host distribution scheme is shown in Figure 3:

Figure 3: Bastion Host Distribution Architecture

Make controls rdist, which is implemented to run over SSH. The combination of Make, rdist, and SSH work very well to allow the centralized management of remote UNIX servers.

When used to manage the remote servers, Make causes several operations to occur. It builds the distribution files, which our network installation process uses. It warns us about which files would be replaced or added on each server being targeted. And finally, it pushes the changes out.

Our network installation process allows us to build new servers very easily. A boot floppy is all that is needed to create or rebuild any of our managed servers. To accomplish this, there must already be several tarred and gzipped *build* files present on the central management server. These *build* files contain the latest server configurations and must be rebuilt whenever a change is made in the corresponding configuration directory tree. During the process of engineering a new set of configurations, we try to minimize the time required to build these files. Putting each set of configurations into its own directory and having a Makefile at the top of that directory forces only those directory trees containing changes to have their corresponding build files remade.

To minimize the number of files pushed out to any given host, rdist compares the sizes and timestamps of the target files. Only those files needing to be updated will cause a file to be replaced on the target server.

When we make a change, we carefully change a central distribution tree and then run a distribution check to verify what we are about to change. The check is done with a "make check" command. This check does two very important things. It allows one to verify that right things were made in the distribution tree. It also points out any other changes that have been made either to the distribution tree or to the bastion hosts. It is possible a colleague is working on one of the bastion hosts or someone is making changes that should not be made. Whatever the case, all changes to be made should be understood before they are implemented. We implement this check by invoking rdist with an option that just prints what must change.

When changes are well understood, they are installed with a "make install" command. This command is almost the same as the "make" check command, except that the Makefile calls upon rdist to actually distribute changes rather than just verify and report changes.

Sometimes special things need to happen before distribution, such as running M4 to generate a new sendmail.cf file. And sometimes things need to be distributed in a special way, such as when sending different versions of password files to different sites. These customizations are easily accommodated in the distribution tree's Makefile and Distfile.

We gain enormous flexibility by using make and rdist, and yet we're still able to maintain a simple and consistent interface throughout our entire distribution mechanism. This makes our distribution system powerful while being easy to use.

3.3 Managing Router and Bastion Host Configuration Files

The way we manage configuration files lets us back out change if we need to. Back-outs to changes are implemented in the same way as normal changes. A previous version of a file is re-installed into a distribution tree, then it is distributed just like any other file would be.

We often use RCS to assist us in coming up with the previous versions of files. RCS gives us past versions of files as well as their history information. RCS directories and files may be embedded into the

distribution trees where they are easy to get at and use. They are automatically ignored by the distribution mechanism, in particular by using

except_pat(.*,v RCS)

in the rdist's Distfile. This minimizes the amount of files copied and keeps configuration distribution as fast as possible.

4. Experiences with the Make Approach

For the most part, using Make to control and manage our configurations has allowed us to overcome the maintenance challenges of our environment. We manage all of our bastion host configuration and router configurations from a few central systems. Management of configurations has become tremendously faster and more consistent. We no longer have to guess what files we need to install on systems after something has changed. When files need to be installed, we no longer have to copy individual files to individual systems. Instead, a command like "make install" automatically figures out what files have been changed, installs those files, and runs whatever utilities are necessary to configure the end system. With our router management, we no longer have to individually log into routers and type in configuration commands or download new configurations.

One of the greatest advantages to this system of ACL maintenance is that the individual files can be put under a revision control system such as SCCS or RCS. (We use RCS). Each change made can be noted in comments and also noted in the version control log. Keeping this history allows us quickly to answer questions such as when various changes were or what was changed on a particular date. It also allows us to compare older versions of the files so that changes can be tracked easily.

We have implemented macros to save typing. Intel is not fortunate enough to have a Class A. Instead, we have many Class Bs and Class Cs. Additionally, we are making increased use of private IP space [5]. We have several gateways that provide secure communications between Intel and other companies. This means that our access-lists from Intel into the DMZs cannot use "permit any" (0.0.0.0 255.255.255.255) because that would allow transit traffic from our business partners through our Internet connections. To prevent this, we have a list of about twenty networks which we have to allow into our DMZs. Instead of coding these into each ACL and having to change each ACL when a new network comes up, we maintain a list of these networks that are incorporated into the ACLs via a macro. The list is in a file called *intel-nets*. The macro expands the variable INTEL_NETS by running a PERL program that takes the appropriate line of the ACL and replaces the INTEL_NETS variable with each network in the intel-nets file.

Our Make driven maintenance has allowed us to scale up the number of firewall complexes at Intel. Our group originally only managed one firewall complex. Now we manage eight and are adding even more. Only with a tool like Make could we manage to scale the number of firewalls with the constraints we have.

The design has proven to be quite robust. We have a commitment to provide 99.95% connectivity to www.intel.com. We can do this because our ACLs allow traffic to fail-over from one gateway to another. We can also lose an entire Internet gateway (both providers or firewall routers) and still get Intel's traffic out to the Internet and back from the Internet in a way we're sure is secure. Yes, ACLs are long, but since they're optimized, most traffic gets through the routers without encountering long processing times.

Our Make based distribution system is a very powerful tool. It can distribute problem solving router and bastion host configurations all across the world. It can also distribute problem-creating configurations all across the world! In any case, this tool requires great discipline.

One problem we occasionally face is when one of our engineers is making changes on a bastion host, but not to the distribution tree on the central server. There can be some very good reasons for this, such as when new software is being tested. It is still a problem because it makes it hard to maintain consistency between the distribution tree and the actual files on the servers. The solution here is always to have staff members check, confirm, and resolve any problems before actually distribute changes. If a bastion host needs to be different for an extended period of time, it should be pulled out of the distribution tree(s) and possibly added to new distribution trees. The powerful nature of this distribution system makes it important to maintain consistency from day to day so each new change receives as much focus as possible.

Being able to quickly and easily push out changes can be a problem. Incorrect changes can be pushed out

everywhere on all bastion hosts and routers, resulting in havoc. Before any change of significance is widely implemented, it is important...CRUCIAL...to make sure that it works beforehand. Disciplined preparation is required before you "just type make."

Changes, especially new versions of packages, should be pilot tested on one bastion host before they are rolled out to all bastion hosts. A new distribution tree should be created when installing a new package. A pilot server can be pulled out of the old package tree's distribution list and added to the new tree's distribution. As the pilot progresses, servers can be migrated from the old to the new tree.

Currently, educational use and charity use of the secure shell protocol is allowed free of charge. However, commercial use requires that you license each server. Have your lawyers visit http://www.ssh.fi/sshprotocols2/licensing/ssh2_non-commerical_licensing.html. Additionally, SSH can use any of several encryption ciphers, including idea, des, 3des, rc4, and blowfish. It can also use RSA public-private key pairs. There may be additional restrictions on the use of these ciphers and authentication methods depending on location.

We have occasionally encountered problems with Make, in that the Makefile itself will have the "dreaded spaces rather than tabs" problem. Sometimes we will find a space at the end of a line, but these are usually easy to diagnose. We have also changed the Makefile without touching any of the files and been told that targets are up-to-date. A little "touch" here and there solves that particular error.

5. Future Plans

We plan a number of extensions to our make driven update system, for both router configuration and host configuration. With router configuration, we currently have a way to optimize our router access lists, but this technique is manual. We would like a way to do this automatically – feeding the ACL usage data into a program and having optimized ACLs emerge, along with a list of ACL entries that can be deleted. We would like to be able to handle cisco's named access list feature. Currently, our scripts cannot deal with named access lists, only numbered. Also, we would like to generate the entire router configuration (not just the ACLs) from a template. That would allow us much greater standardization of router configurations and make it faster to bring up new gateways.

Another item that we wish to improve is the way we change our router configurations. Currently, our scripts log into the same way as a network administrator would – telnet to the router, get enable privileges, make changes, and then log off. Clearly this approach has problems as telnet sessions traverse DMZs. Using enhanced security features like Kerberos should improve the security of our router change processes.

With host configuration, we mentioned how it was easy to push out an error-ridden configuration to all of our bastion hosts. We would like to implement some sanity checks that would check configurations for common errors before they were pushed out and then run immediate checks after they are installed. This is particularly important when maintaining DNS tables. This way, we would notice if we pushed out a change that somehow damaged key DNS records at Intel.

6. Conclusions

Our experiences show that Make can be an effective way to maintain and administer firewalls. Intel Corporation manages a number of geographically dispersed firewall complexes that failover between themselves, including both router packet filtering access lists and bastion host configuration, using Make driven configuration. While this approach requires discipline to use (it is just as easy to push out a bad configuration as well as a good configuration), "just typing Make" solves many of the challenges of administering distributed yet interacting firewalls.

References

[1] Chapman, D. Brent and Elizabeth Zwicky. *Building Internet Firewalls.* O'Reilly and Associates, Inc., Sebastopol, CA 1995. pp. 66.

[2] Oram, Andrew and Steve Talbot. *Managing Projects with Make.* O'Reilly and Associates, Inc., Sebastopol, CA 1991.

[3] Libes, Don. *Exploring Expect.* O'Reilly and Associates, Inc. ., Sebastopol, CA 1995.

[4] SSH Home page. http://www.ssh.fi/

[5] Rekhter, Y., B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear. "Address Allocation for Private Internets." RFC 1918, February 1996.

Tricks You Can Do If Your Firewall Is a Bridge

Thomas A. Limoncelli
Lucent Technologies, Bell Labs
Murray Hill, NJ, 07974
http://www.bell-labs.com/user/tal
tal@bell-labs.com

Abstract

Firewalls that forward packets like a bridge, rather than as a router, have many operational benefits. By decoupling routing from filtering, the firewall becomes a pure filter, unburdened by routing table or interface configuration. The result is increased flexibility. This paper explores some of the benefits we have found. Most of the benefits stem from the fact that a bridged firewall requires fewer transit subnets. Sometimes transit subnets are completely eliminated. It can be placed between any two network devices and act like a line filter without needing to change the logical routing of the network. It is easy to put one in series with another firewall for testing. Our examples include replacing an old firewall with a new one, moving a firewall from one router to another with zero downtime, firewalling off an individual office or lab, and others. In many cases topology changes are made without service interruptions. The operational procedures become much more simple. The paper also suggests future directions for research in this area.

1 Introduction

Firewalls filter packets by sitting between two network points and deciding whether or not to pass each packet based on a set of rules. Sitting between two points requires the device to pass packets like a router (a Layer 3 device) or a bridge (a Layer 2 device). The fact that a firewall is bridge-like or router-like is not usually emphasized by vendors because the same firewall features can be provided either way. However, we have found interesting operational advantages to bridge-like firewalls.

2 Background and history

A firewall is a device that filters TCP/IP packets based on a set of rules. As each packet passes through the system, the rules are processed to determine a "pass" (forward the packet) or "no pass" (drop the packet) decision. Depending on the security policy required, different kinds of rules may be constructed. For a general discussion about firewalls refer to [Cheswick94] or [Chapman].

Many of the early firewalls were routers modified to perform certain filtering functions. As firewalls became more complicated, vendors began using UNIX workstations with two network interfaces. The workstation would be configured to route packets between its interfaces and in some cases would run a modified kernel and software that was capable of performing some kind of filtering or proxying functionality. Later "standalone firewalls" became popular because of their simplicity and the advantages that network "appliances" have.

In these early generations, the device connects two IP networks and therefore must also contain routing functionality to know how to forward packets. They must be configured with routing tables that describe which IP subnets are where. These routing tables are usually configured statically rather that relying on potentially insecure dynamic protocols.

A few new firewalls such as the Lucent Managed Firewall [LMF] and others forward packets like a bridge. That is, they act as a learning bridge between two devices. These two devices are usually routers which are much more capable of performing the complicated routing tasks required in modern networks. By decoupling routing and filtering into separate boxes, each can focus on one task and

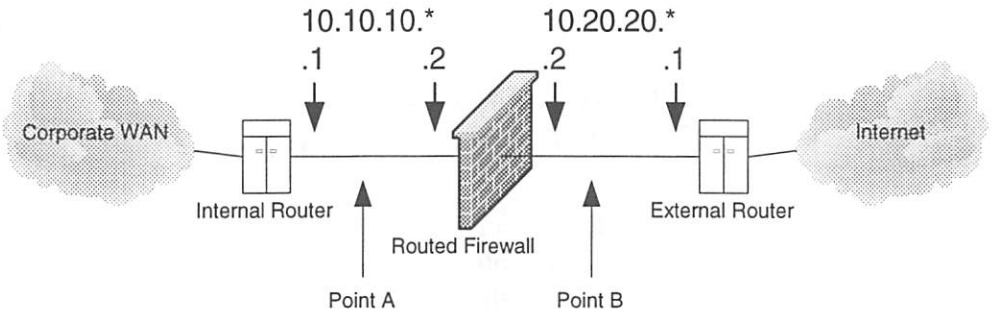


Figure 1: Router-like firewalls require two transit IP subnets.

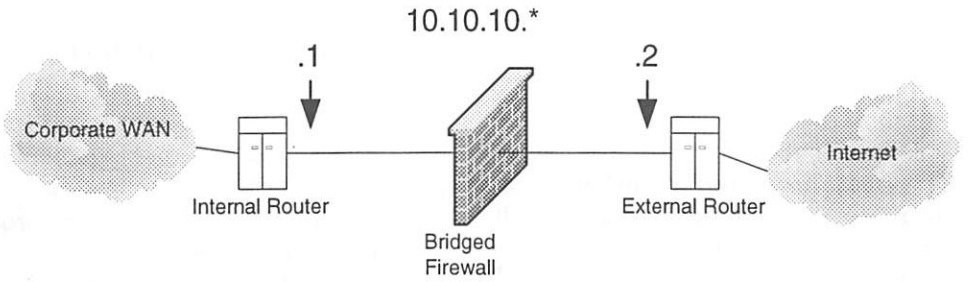


Figure 2: Bridge-like firewalls require no additional transit IP subnets.

do a better job of it. The firewall still bases its "pass/no pass" decisions for each packet based on the Layer 3 information within the packet. Even though the term "bridge" is used, it does not make its decisions based on the L2 ("MAC") addresses in the packet. If we were to compare the ruleset of a bridged firewall vs. a routed firewall we should find the exact same rules if they implement the same policy.

I will use the phrase "routed firewall" to mean a firewall that passes packets the same way that a router does, as a L3 device. That is, it routes packets between two IP subnets. It is not necessarily meant to imply that an off-the-shelf router being used to filter packets. Similarly, I will use the phrase "bridged firewall" when I mean a firewall that passes packets like a bridge or other L2 device, not necessarily an off-the-shelf bridge that has been configured to filter packets.

3 Transit subnets

Most of the techniques in this paper rely on the fact that a bridged firewall can be inserted between two points without changing the IP routing topology of the network. Without a firewall it only takes one "transit" subnet to connect two devices. To insert a router between two network elements one must use an additional transit subnet.

Figure 1 shows that two small transit subnets are required when a routed firewall is between two routers. Figure 2 shows how only one transit subnet is required when a bridge-like firewall is used.

The first benefit of reducing the number of transit subnets is a reduction in the use of IP address space. Allocating one fewer subnet allows you to use that address space elsewhere. Conserving IP address space is important. The second benefit is less obvious. When adding an additional transit subnet one must reconfigure one device to use the new subnet. This interface configuration and the routing table adjustments is an additional burden on the installer. With a bridged firewall the two network devices need not be reconfigured. This latter benefit is the basis for much of what is described in the remainder of this paper.

4 Trick: Ease of deployment

Replacing our old (routed) firewall with our new LMF firewall was made considerably easier because the LMF is a bridged firewall. Because the new firewall was a bridge, it was programmed with its filters and then was tested in various positions in our network with limited disruptions. Backing out would have been quick and easy if problems had been discovered.

Our previous firewall was named "`stile`."[1] It was a prototype that used stateful inspection filtering[2] like the firewall that was replacing it. However, it was a routed firewall.

We were very cautious when we deployed the LMF. It was a prototype and not even an announced product at the time. We were testing it in an environment with hundreds of users that would be very unhappy to lose Internet service.

The initial test of the LMF prototype was done by putting it in series with the old firewall. Our firewall network configuration was similar to Figure 1. First the LMF was programmed with the same filter rules as `stile`. The LMF was then installed behind the old firewall, at Point A in Figure 1. Now all traffic was filtered twice, but if any problems were discovered with the LMF, the problem would be inside the zone protected by the firewall.

Since the same developers had created both prototypes, the logs that each generated were extremely similar, or similar enough that simple pre-processing via `awk` should produce output that could be compared with utilities such as `diff`. Intrusion attempts would be seen in the `stile` logs but not the LMF prototype logs, otherwise they should be the same. The LMF prototype's logs should not include any entries that were not in `stile`'s logs. If this did happen, it would be a sign that the LMF was not following the rules the same way. The LMF prototype logs should also not be missing any entries from `stile` with the exception previously mentioned.

Installing a bridge is a matter of two quick cable changes that takes seconds. No interfaces needed to be reconfigured and no routing tables had to be adjusted. If the LMF were a routing firewall, inserting it into the path would have taken more effort and would have involved a longer outage. This would have slowed down insertion and (more importantly) the possible removal that would have followed if we discovered the LMF prototype wasn't working properly.

Turning a multi-minute outage into a few short seconds is important because users did not feel an outage. A 3-second pause when accessing the Internet is not noticed by most users. Modifying router configurations to insert the firewall would have been a serious disruption.

Similar time would be required if the device had to be removed. If a problem was discovered, being able to rapidly remove the device is appreciated.

Once validation was completed, it was time for a more serious test. We moved the LMF prototype outside the old firewall, at Point B in Figure 1.

We did similar log comparisons. This time, only the LMF logs should show any intrusion attempts and the remaining log entries should be similar (after pre-processing).

Moving the LMF to position B was also a matter of a few quick cable changes. The same LMF, with no changes to its configuration, was used. No routing tables had to be changed on the LMF, `stile`, or the other routers. Again, if these tests found problems with the prototype, it could be removed quickly. In fact, it could even be removed by operational staff with minimal training rather than requiring network or security administrators.

Once the LMF was validated, removing `stile` was relatively time consuming because it involved removing a transit subnet and the making the appropriate routing table and interface updates. Luckily that would be the last time we would have to deal with a routed firewall!

Later when the LMF prototype was replaced with the actual LMF product, we repeated the same process of placing the new firewall behind, then in front of, its predecessor. However, now removal of the old model didn't require any routing changes because we were replacing a bridge with a bridge. Removing the prototype was a snap.

[1] a stile is a ladder or set of steps that goes over a fence.

[2] Stateful inspection is a filtering technique where packets are examined in the context of the entire session. For example, rather than permitting all telnet packets through, the firewall maintains a list of which telnet session have been initiated, and only permits telnet packets associated with those sessions. For more details, see [Chapman].

Now we have enough confidence in the product to reduce the testing. As we receive new versions of the product we only test it by installing it outside of the current firewall. This saves a lot of time. We currently have two complete sets of hardware. One tests the "even releases" and the other tests the "odd releases." As a result, we can switch back and forth rapidly with very little disruption. With a routed firewall the interruptions would be much larger and we would lose our flexibility.

It should be noted that sometimes the LMF must be rebooted when it is moved. As a learning bridge, it learns which ethernet (MAC) addresses are on each side. If a MAC address moves from one interface to the other, it is blocked. The theory here is that machines usually don't physically move around your network. The LMF does not time out a MAC address like a real bridge, the only time the table is cleared is when the device is rebooted. While none of the steps described in this paper require a reboot, other experiments we performed did require a reboot. If you plan on doing your own experiments please remember this. There are other bridged firewalls on the market that may or may not work the same way. Contact your vendor for more information.

5 Trick: Moving to a new router

We were able to move the firewall from one router to another without any downtime, without any changes to the firewall, and (until the very end), without bothering the owner of the first router.

Figure 3 is a simplified diagram of how our routers and firewall are connected for our LAN. Our LAN involves a cluster of routers connected at a central backbone. Each router serves a separate customer group and in many cases is owned and controlled by that customer group. The firewall is attached to one of the routers. This means that one group was slightly "closer" to the Internet than others.

Each of our internal routers contains a dynamic routing table that contains all the subnets ("prefixes") within our corporation. The default route on each router points towards the firewall (either to the external router or to the router attached to the firewall). The assumption is that if a destination host is unreachable via our internal route tables, it

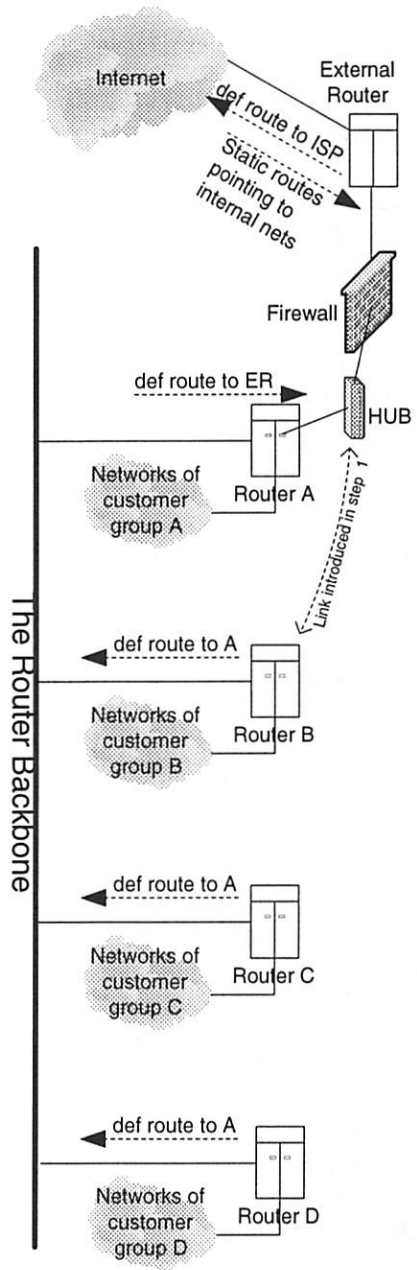


Figure 3: Our router inter-connections

must be on the open Internet. This is more efficient than having our internal routers maintain the entire corporate routing table as well as the Internet's route list. In Figure 3, the firewall is connected to Router A. Router A's default route points to the external router. The remaining routers' default routes point to Router A.

Due to operational issues, we needed to move the firewall off Router A onto Router B.

The move consisted of a sequence of carefully planned steps. After each step, we ran a battery of tests to verify our work. The tests consisted of sending packets from a host in each customer group to a number of places on the internal network, our extranet, and on the open Internet. The test was automated into a simple shell script that completed in a matter of seconds.

One more point must be understood by the reader before continuing. IP routes can be asymmetrical. That is, the path a packet takes to get from host X to Y may be different than the return path. The difference in the two paths might be slight or huge. The hosts do not care. During the following steps, there will be times when outgoing packets take a different route than incoming packets. Keep this in mind.

The first step was to achieve physical connectivity between Router B and the hub. Since these devices were in very different parts of the building, this would verify a series of cables, jumpers, and wire-fiber adaptors. Once physical connectivity was achieved, we configured the interface on Router B and verified that we could ping from Router B to Router A. The battery of tests was run as a baseline. At this point nothing should have been different. Packets from Router B to the Internet would still go through the Router Backbone, to Router A, through the firewall, to the external router.

The second step was to have outgoing packets from Router B go to the external router. Incoming packets would still come in via Router A. This step was achieved by changing the default route on Router B to point to the External Router (though the firewall). We re-ran the battery of tests and everything was fine. There was no service interruption due to this step.

At this point we took a short break. We waited to see if any users in Customer Group B reported any problems. Would they notice that their outgoing packets were taking one fewer hop than incoming packets? Nobody noticed anything and no service problems were reported. We could continue.

The third step was to configure more routers to send their outgoing packets via Router B. We changed the default route on Router C and D. We continued running our battery of tests. Again, no problems, no service interruptions, and nobody noticed that their packets were going out via Router B but returning via Router A.

Step four focused on the incoming packets. The External Router has a static route for every internal network, each pointing to Router A. All other networks are routed towards our ISPs. In this step, we change the static routes to direct packets to Router B instead. By doing this, we now had symmetrical routes for all the customer groups except for group A which is still sending packets out directly to the External Route via the firewall. Again, the tests were run. Again no service interruptions were caused by this step.

The fifth and final step was to contact the owner of Router A and request that the default route be changed to point to Router B so that it is the same as all the others routers (except Router B itself). The owner was incredulous at what we had done but agreed to make the one change required of him. Once the default route of Router A was pointing at Router B, all traffic was flowing symmetrically and was going to the firewall via Router B.

The final step was to disconnect the cable between Router A and the hub. Before we did this we ran various `traceroutes` to make sure we were disconnecting a cable that had no active traffic.

These changes were made without scheduled downtime. At some point packets were traveling in asymmetric paths, but users wouldn't notice this. Also, the stateful inspection process of the firewall was not disturbed by our live changes.

Table 1 depicts the route table settings after each step of the process. The columns for Router A–D show where the default route pointed. The last column shows how the External Router's static routes for internal subnets were directed. An asterisk ("*") denotes that packets for that customer group were routed asymmetrically at this stage.

	A	B	C	D	ER
At start/Step 1	ER	A	A	A	A
After Step 2	ER	ER*	A	A	A
After Step 3	ER	ER*	B*	B*	A
After Step 4	ER*	ER	B	B	B
After Step 5	B	ER	B	B	B

Table 1: Route table settings after each step.

This technique could have been done with a routed firewall but it would have required service interruptions and would have been much more labor intensive, as each step would have required routing table and interface configuration changes.

6 Trick: Firewalling off my office

With a firewall that is a bridge, I can firewall off my office without the network administrators knowing or caring. I simply place it between the network jack in my office and my workstation. Because it is a passive filter, no routing changes are required.

If my firewall was a router, I would have to request a dedicated subnet, have a dedicated router port with dedicated connection to my office, deal with routing issues, etc. It wouldn't be nearly as fun.

We often isolate an office with a learning bridge or etherswitch to provide more bandwidth. Simply connect one port of the bridge to your office's network jack and another port to what used to be plugged into the network jack. Doing this with a bridged firewall is very similar, except this bridge can have a complicated security policy. If I use a ethernet hub, I can even have more than one machine in my office. This creates a many-to-many relationship within the same subnet. This can not be done with a routed firewall.

The security policy that I used was quite simple. TCP connections from my machines could connect to anyplace else. Incoming TCP connections could only come from certain machines on certain ports. For example, I decided that only one particular machine could `telnet` to me, and a larger list of machines could reach me via `ssh`. UDP protocols such as DNS and the like had to be decided on a per-protocol basis. I started with a very strict security policy and added exceptions as I discovered what services I needed (i.e. what broke due to my strict security policy). Reviewing the policy violation log on the firewall was helpful to find out what protocols I needed to add to the policy to regain functionality.

This was much easier than constructing the policies for the firewall that connects us to the Internet. For example, there was no need to set up internal and external SMTP gateways to funnel mail though known "safe" software. DNS did not need to be passed though something like `dnsproxy` [Cheswick96] as I was fairly confident in the quality of the DNS data I was receiving. (And if I had been wrong, chances are other machines would fall to the attack before mine). Another difference is that one would never pass NFS through a firewall to the open Internet. However, in this scenario, I could permit my machines to mount from file servers outside my office and live with the risks.

Since I can have more than one machine on either side of a bridge, this technique could also be used to easily firewall off an entire cluster of machines, say, in a lab.

I could also develop rules that were a little more sneaky. For example, I could hide my machines from the subset of machines used by my boss. I could also configure the firewall to log, but not reject, sessions from particular groups of machines if I suspect they are probing me. However, since my firewall is transparent, they would not be able to see that I am monitoring for their packets.

The real benefit gained from the firewall being a bridge is that I was able to do this without any awareness or adding to the workload of our network administrators. If this were a routed firewall, connecting the dedicated transit network to my office would have involved a long wait as changes were made in wiring closets, etc.

7 Trick: Hiding machines

Inspired by the ability to do the previous trick without the involvement or knowledge of the network administrator[3] I thought there might be some other interesting things I could do without the network administrator's knowledge. For example, I could steal IP addresses. Normally using the IP address of your own choosing is dangerous and the author

[3] I'm exaggerating. I am the network administrator!

recommends you only use IP addresses allocated by a centralized authority. If you use a random IP address that is later allocated to someone else on the same subnet, the address collision will make both machines unusable.

However, with a bridged firewall, filters can be constructed so that nobody outside your firewall will ever see packets from machines using the stolen IP addresses. Of course, this means that these machines can not talk with any servers outside of your firewall, or even the default route for the network.

The DNS data for your hidden machines would be that of the hosts whose stolen IP addresses you are using. At least one machine would have to use a non-stolen IP address so that it could gateway or relay for the others. This machine would have to provide services such as DNS, SMTP, DHCP, web proxy, and other protocols. Clients could conceivably have all the services they need either locally (compute and file service) or via proxies on the gateway machines. In fact, with `dnsproxy` you could even fake the DNS data so that the machines inside your little kingdom see the DNS data that you want them to see.

Those proxy services aren't needed if you do not interact much with the outside world. Then again, if you don't need connectivity to the rest of the world, you don't need a firewall, you can just pull the plug. That doesn't require any intervention from your network administrator either. In this case one may want to use the special IP space reserved in [RFC1597].

This is a lot of work to go through to hide a couple machines. Certainly this is more work than getting a transit network connection set up to your router. Unless you have a real vendetta against your network administrator or are James Bond, I don't think this is very useful besides an interesting theoretical discussion.

8 Trick: Firewalling off a lab

Firewalling off an office is interesting but in the future we plan on taking this idea even further. If we can firewall off an office, can't we firewall off an entire division?

The challenge here is that a large group of machines usually involves many subnets. Things become easier if one can pass a routing protocol through a firewall. A routed firewall could not pass a routing protocol like OSPF without re-implementing the entire routing protocol. That would be very difficult.

A bridged firewall could do this more easily by just blindly passing all packets from your chosen routing protocol and letting the surrounding routers process the packets, something they should be very good at. There is a loss of security because we are now depending on the routers to properly process the packets. We wouldn't be protected against routing attacks. However, we're already inside the main firewall, so we can take that risk. A routing attack could be thwarted by our firewall rules.

There are two unsolved problems we face when firewalling off a group of machines as large as a division. One is technical, the other is policy. The technical issue is that the network of a very large division usually has multiple connections to the rest of the company. In this situation we would need to put a firewall at each entry. If there are more than one connections to the division asymmetrical routing may be involved: a packet may enter via one connection and the reply might leave via another. Dealing with asymmetric routes through two different gateways requires the rapid sharing of state information. There are performance and security issues to getting this right. There aren't any products that do this today. This may change as demand for it increases.

The other problem is simply policy. Before one can define a set of firewall rules, one must determine the security policy. That is, you must define "what are you trying to protect?" and "how much risk are you willing to take?" So far, we have not been able to determine if, in our particular situation, we are protecting ourselves from the rest of the company or are we protecting the rest of the company from us. There are many arguments for both. For example, my division is very research focused. We try new protocols and take different risks than the rest of the company. Are we trying to protect the rest of the company from the risks we take?

On the other hand, there is an excellent argument that states we should be protecting ourselves from the rest of the company. An internal network census found 260,000 live hosts corporate-wide. When the Internet was 260,000 hosts large, we had a fire-

wall to protect us from it. If our internal corporate network is that large, should we be considering a firewall to protect us from it? What's the difference between 260,000 Internet hosts run by people that we've never met, and 260,000 corporate hosts run by people that we've never met, but have the same source of paychecks? What's to say that one of our 150,000 employees hasn't accidentally (or on purpose) given access to outsiders? Who's to say one of our local users hasn't accidentally done the same?

In the future we hope to investigate these issues.

9 Trick: Connecting directly to the backbone

We put our firewall directly on our router backbone, making it zero additional hops from each customer group. Previously, all but one customer group was an extra hop from the firewall. This change was not done for performance reasons as the additional hop would go unnoticed in today's huge Internet. Instead, this was done to prevent the one router from being a single point of failure. We do not want an outage in one customer group to cause outages in other customer groups.

Astute readers will point out that the firewall itself is a single point of failure. We deal with that in other ways (such as a hot spare, etc.).

To achieve this goal we will perform similar steps to when we moved the firewall from Router A to Router B (see Figure 3).

First we connect the Router Backbone to the hub. We assign a secondary IP address to the external router so that the same interface is both on the hub's IP subnet and the Router Backbone's IP subnet. The same battery of tests is used as before.

The next step is to configure outgoing packets to go directly to the External Router. The default route of Router A, B, C and D is changed to point to the External Router's backbone address. Now all outgoing packets are going directly to the External Router after, of course, being filtered by the firewall.

Now let's take care of the incoming packets. The external router must be configured with a static route for every subnet to the appropriate customer groups' router. We can save some time by using a default route for the router with the most routes and providing static routes for all the other routers.

Finally we can remove the secondary IP addresses that enable the hub to be both on the Router Backbone and the old transit net simultaneously. Before we do this, we can do a final test to make sure everything was done right. We can disconnect the wire between Router B and the hub. If our battery of tests succeeds, then we know it is safe to remove the old secondary IP addresses.

It is possible to do this with a routed firewall, but downtime would be required as we made the routing changes. Many routed firewalls do not support secondary IP addresses, which means the changes could not be done live without a service interruption. This is an excellent example of one of the benefits of decoupling routing from the firewall. A full-featured, dedicated router is more likely to have the routing features we need than a routed firewall.

10 Conclusions

Bridged firewalls decouple routing from filtering. This provides interesting operational benefits.

As a bridge, it can placed between any two network devices. If those two devices are hubs, a subset of machines on the same IP subnet can be firewalled from each other, something a routed firewall could never do.

Its extremely easy to put a bridged firewall in series with other firewalls for testing or other purposes.

With a bridged firewall, many network topology changes can be done without service interruptions. These physical and logical changes are not on the firewall, but on the routers that exist in the network. Thus change is localized and therefore simplified. Fewer modifications on the firewall is also a plus because often security measures to "lock down" a firewall make changes to its configuration bothersome.

In our rapidly changing network, these benefits have been a remarkable advantage. Many of our examples involved a total elimination of service disrup-

tion as major changes were made. Some required disruptions on the order of seconds. We reduced or eliminated the number of transit networks from two to one, or from two to zero.

In the future we hope to explore firewalling off large labs of machines and experiment with what policies are appropriate when firewalling off larger and larger numbers of machines within a corporate intranet.

We also hope to explore the opposite direction: firewalling off extremely small groups of hosts. A bridged firewall might be just the right technology for a firewall in the home, especially given that IP addresses are not allocated in abundance in that market without higher charges.

Being able to place a firewall between any two network devices changes the way we think about firewalls. They are no longer routers with special filtering abilities. They are independent filters that can be put between any two devices, anywhere, at any time, even between large groups of devices (point to point, one to many, and many to many). They can even be put in series with each other. Decoupling routing from filtering lets us take advantage of full-featured, dedicated routers which may have features that a routed firewall may not provide. We have only begun to scratch the surface of the new possibilities introduced by this new paradigm.

11 Acknowledgments

This paper couldn't have been written without the help and support of Bill Cheswick, Lookman Fazal, Paul Glick, Darren Shaw, and everyone involved with the LMF prototype team: David Majette, Mike Coss, and Ron Sharp. Thanks to Josh Simon, Tommy Reingold, Adam Porter, and William LeFebvre for their editing and feedback.

References

[Chapman] "Building Internet Firewalls". D. Brent Chapman and Elizabeth D. Zwicky. O'Reilly and Associates. Cambridge, MA. 1995.

[Cheswick94] "Firewalls and Internet Security; Repelling the Wily Hacker". W. R. Cheswick and S. M. Bellovin. Addison Wesley. Reading, MA. 1994.

[Cheswick96] "A DNS Filter and Switch for Packet-filtering Gateways". W. R. Cheswick and S. M. Bellovin. Proceedings of the 6th UNIX Security Symposium. July 1996.

[LMF] The Lucent Managed Firewall. `http://www.lucent.com/security/`

[RFC1597] "RFC1597: Address Allocation for Private Internets". Y. Rekhter, B. Moskowitz, D. Karrenberg, G. de Groot. March 1994

INVITED TALK

ABSTRACT
The Evolution of VLAN/ELAN Architecture at Vanderbilt University

John J. Brassil

Network Design and Engineering
Vanderbilt University
john.j.brassil@vanderbilt.edu

This talk will examine the design and implementation of VLAN architecture at Vanderbilt University that began as part of the Backbone Reengineering Project (1995-98) and the subsequent changes to that design. Since the backbone is ATM-based and edge networks are Ethernet LANs, the parallel Emulated LAN (ELAN) architecture and its evolution will also be described.

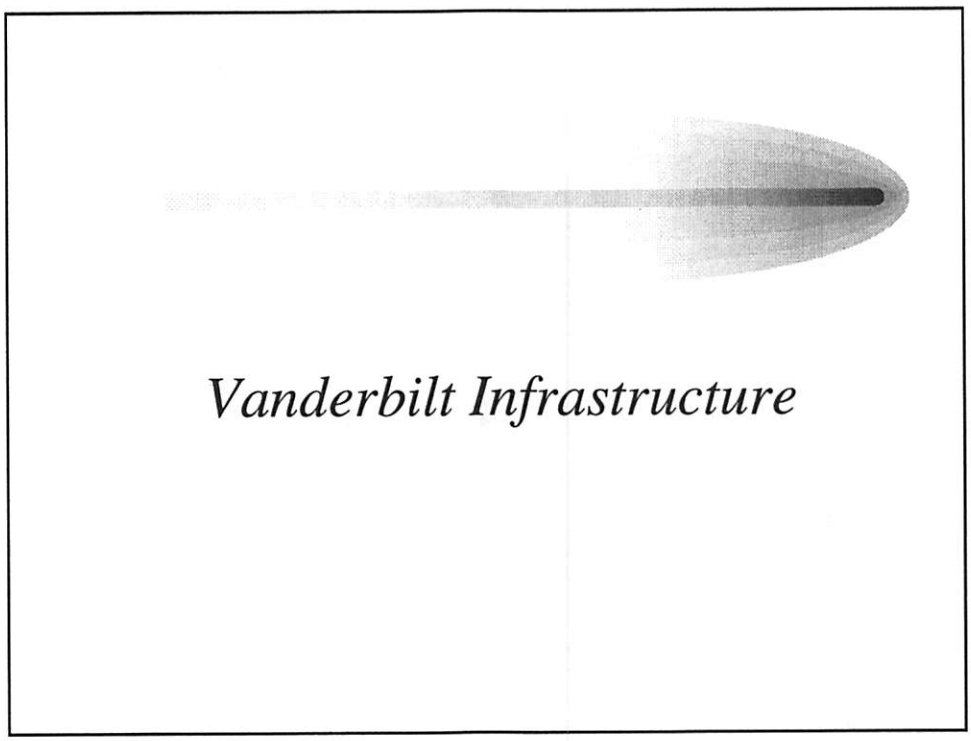
The talk is intended primarily as a case study of VLAN/ELAN implementation in a large university or corporate environment, the factors that influence design decisions, and the tradeoffs/pitfalls that accompany a particular choice. Design considerations for an MPOA (Multi-Protocol Over ATM) architecture will also be discussed.

The Evolution of VLANs at Vanderbilt

John J. Brassil, Network Engineer
ACIS Network Design & Engineering
Vanderbilt University

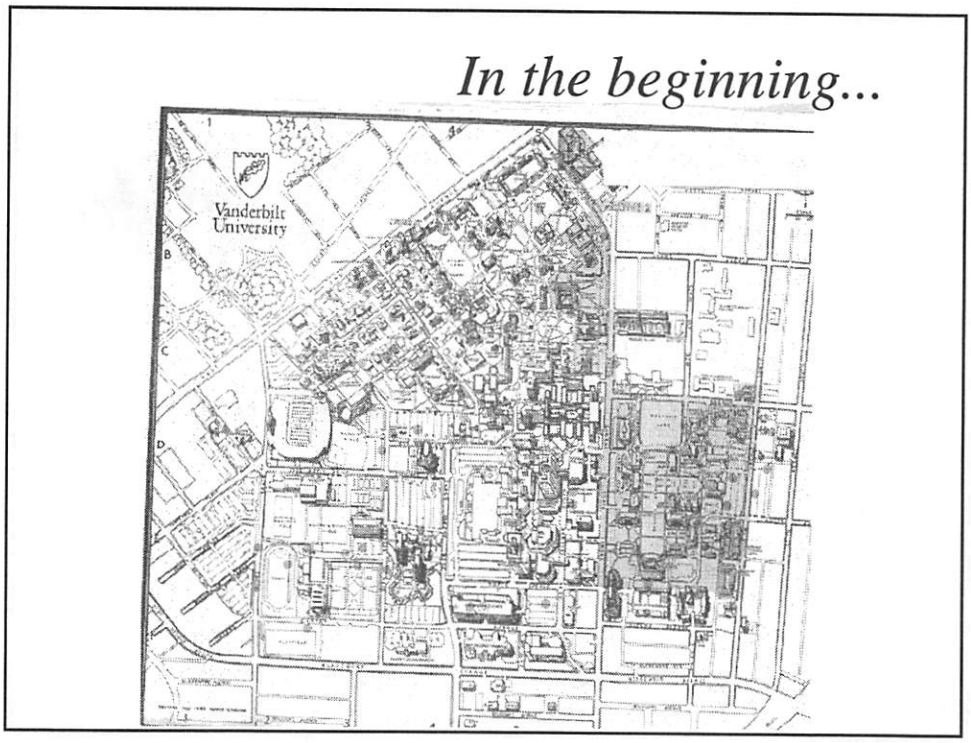
Introduction

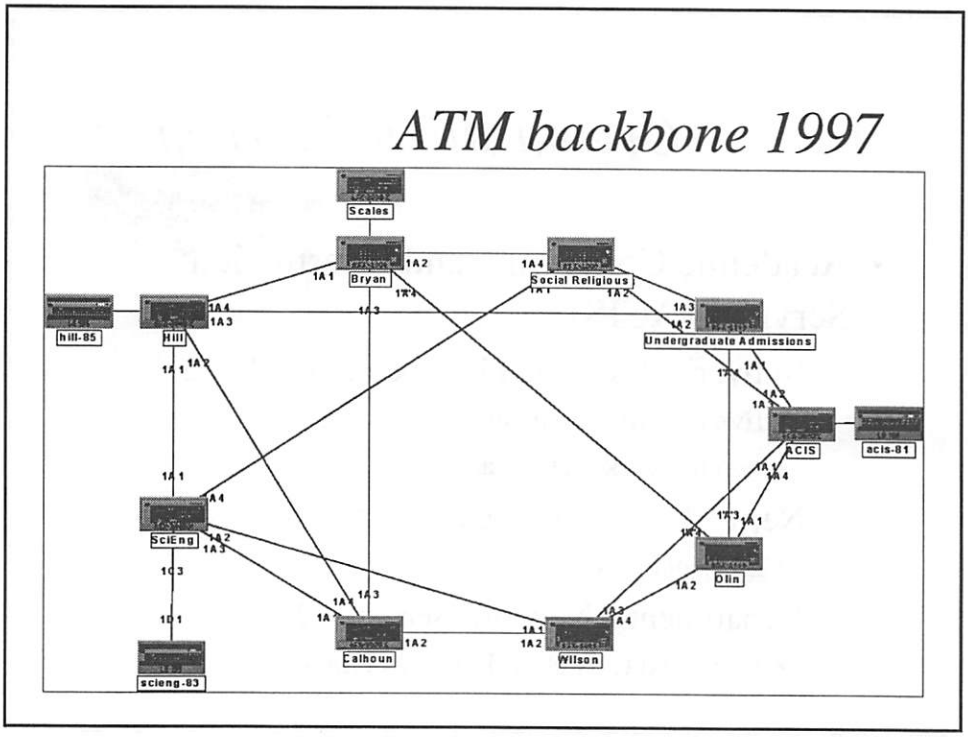
- Vanderbilt Infrastructure
- Technology Overview
  - Broadcast/Collision Domains
  - VLAN vs. ELAN
  - MPOA
- VLAN design refinement

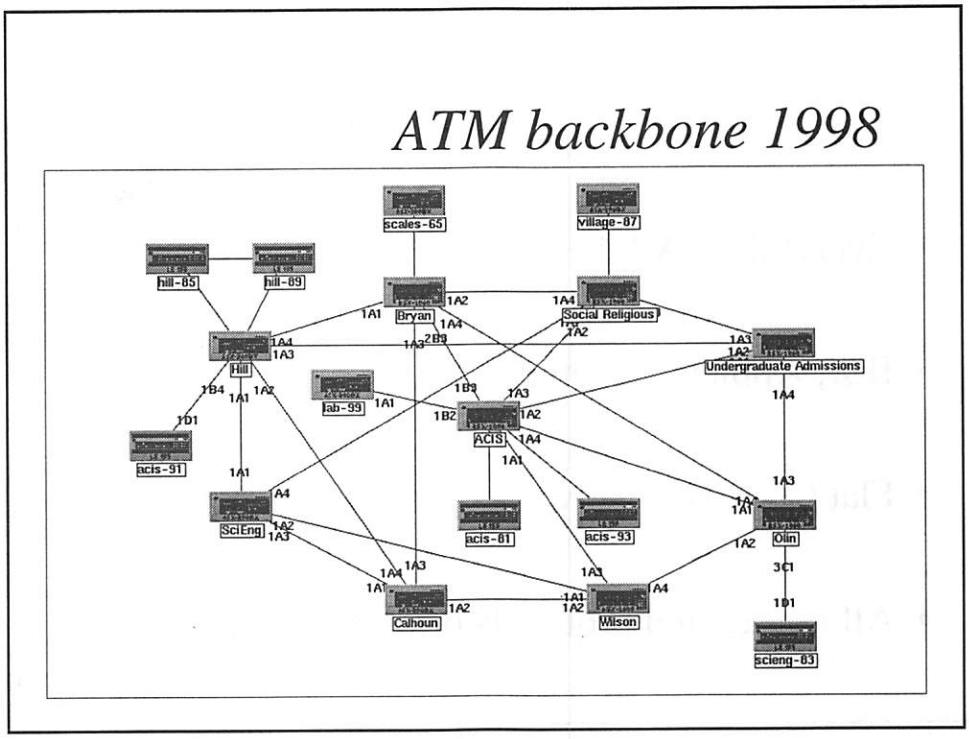


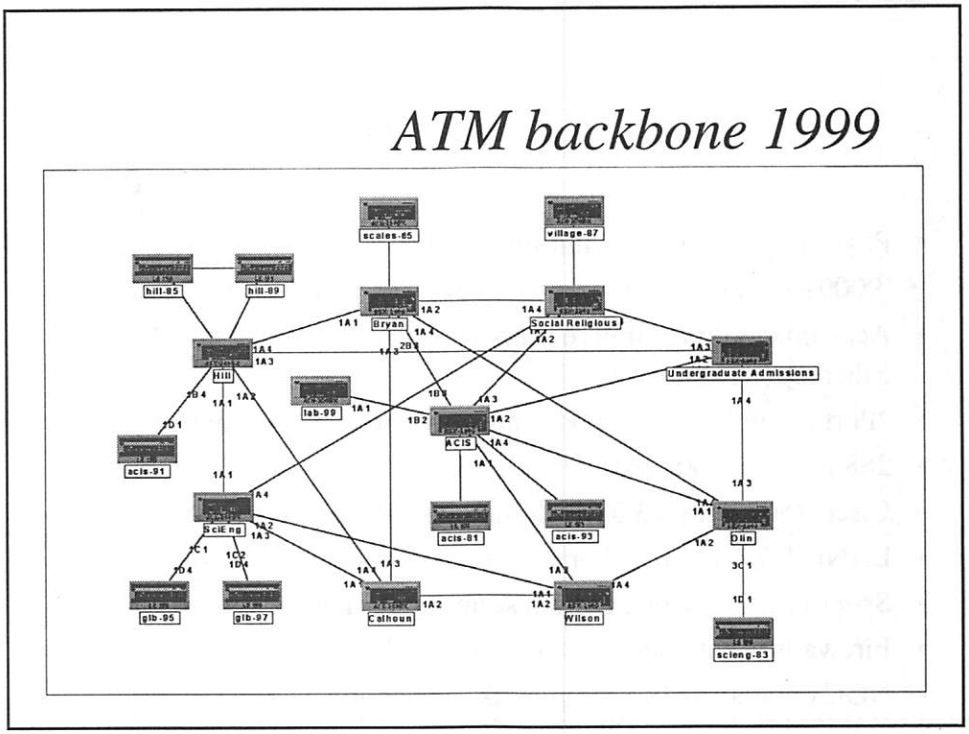
Organizational Structure

- Academic Computing and Information Services (ACIS)
  - Network Design and Engineering (ND&E)
    - five network engineers
    - two network technicians
  - Network Operations Center (NOC)
    - 24x7 operations staff
  - Departmental Network Services (DNS)
    - three desktop networking specialists


In the beginning...
Vanderbilt University


ATM backbone 1997
Scales
Bryan
Social Religious
hill-85
Hill
Undergraduate Admissions
ACIS
acis-81
SciEng
Olin
Calhoun
Wilson
scieng-83


ATM backbone 1998
scales-65
village-87
hill-85
hill-89
Bryan
Social Religious
Hill
Undergraduate Admissions
lab-99
ACIS
acis-91
SciEng
acis-81
acis-93
Olin
Calhoun
Wilson
scieng-83


ATM backbone 1999
scales-65
village-87
hill-85
hill-89
Bryan
Social Religious
Hill
Undergraduate Admissions
lab-99
ACIS
acis-91
SciEng
acis-81
acis-93
Olin
glb-95
glb-97
Calhoun
Wilson
scieng-83

Protocols in Use

- Everything, All the Time
- IPX, Appletalk routed
- Flat Class B IP network (255.255.0.0 netmask)
- All non-routed protocols bridged together

Current Snaphot

- Physical plant: Cat5, Multimode fiber
- 9000+ nodes in over 100 locations (~7K active MACs during day)
- Academic/Administrative users on Switched/Shared 10 Mbps Ethernets with some 100 Mbps
- "Port per pillow" 10Mpbs connections in residence halls
- 288 modems (60/40 33.6/56k Flex)
- Cisco 7507 with 13.5 Mb commodity + OC-3 Internet2 connection
- LANE 1.0 ATM backbone with meshed OC-3 PNNI links
- Spectrum, Foreview, custom scripting for network management
- Firewalls at Internet gateway, HR and Telcom
- NOC coverage 24x7, Help Desk 9-Midnight M-F

Technology Overview

Broadcast/Collision Domains

- Physical Ethernet LANs have a single collision and broadcast domain
- Bridged Ethernets have multiple collision domains
- Routed Ethernets have multiple broadcast domains
- Switched Ethernets have single user collision domains
- VLANS allow Ethernet segments to be physically non-contiguous broadcast domains

VLAN vs. ELAN

- VLANs can be:
  - port based
  - MAC address based
  - protocol based
  - application based
- An ELAN is a type of VLAN that traverses an ATM core

ATM Forum LAN Emulation 1.0

- Also known as "LANE" or "LAN em"
- Separate servers for Configuration, Address Resolution and Broadcasts/Multicasts
- ATM "cloud" treated as multiple broadcast domains (emulated LANs or ELANs)
- Emulated Ethernet or Token Ring

LANE components

- LEC - LAN Emulation Client
- LECS - LAN Emulation Configuration Server
- LES - LAN Emulation Server
- BUS - Broadcast/Unknown Server

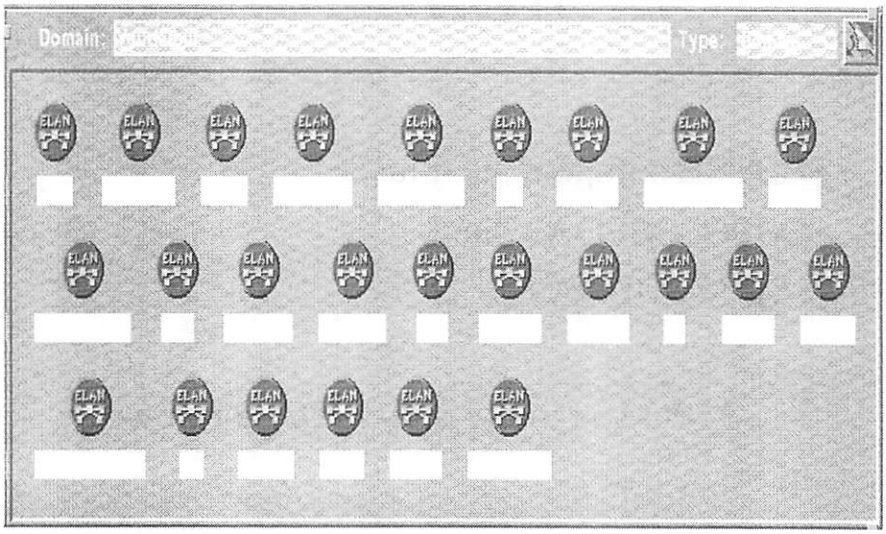
Multi-Protocol over ATM (MPOA) components

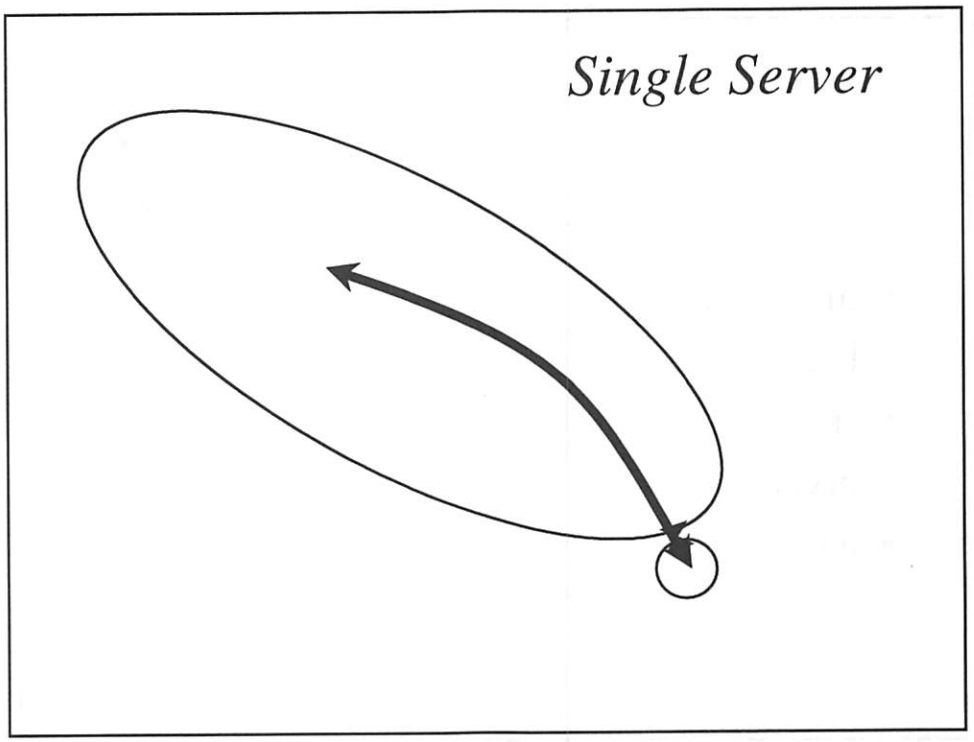
- LANE components, plus:
- MPC - MPOA Client
- MPS - MPOA Server

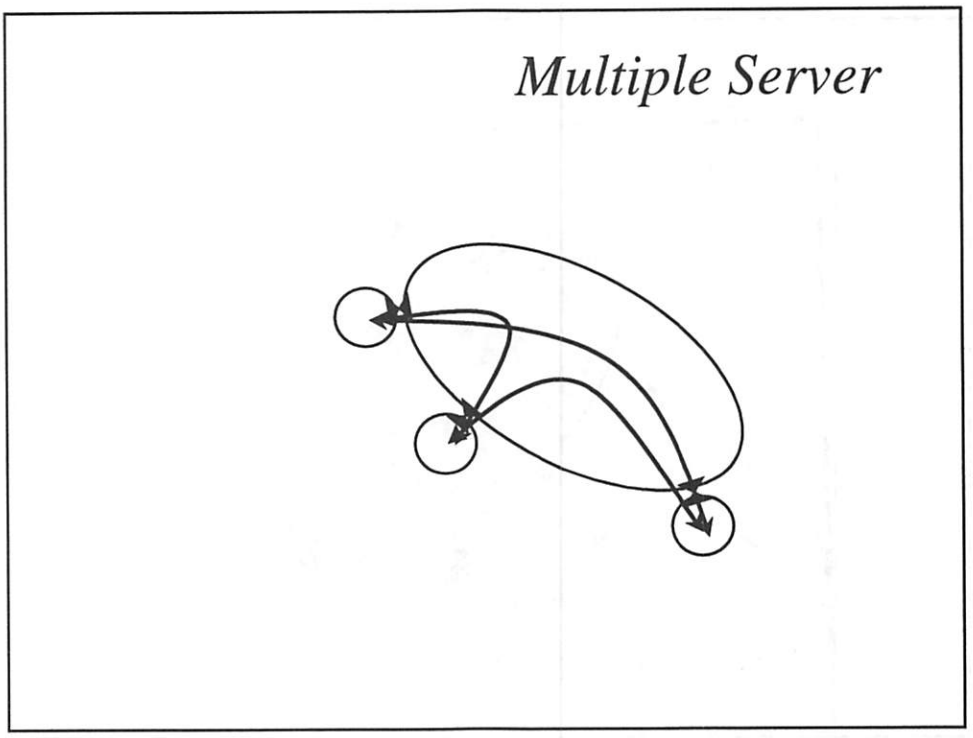
MPOA is really only IPOA for now

VLAN Design Refinement

Current VLAN Architecture



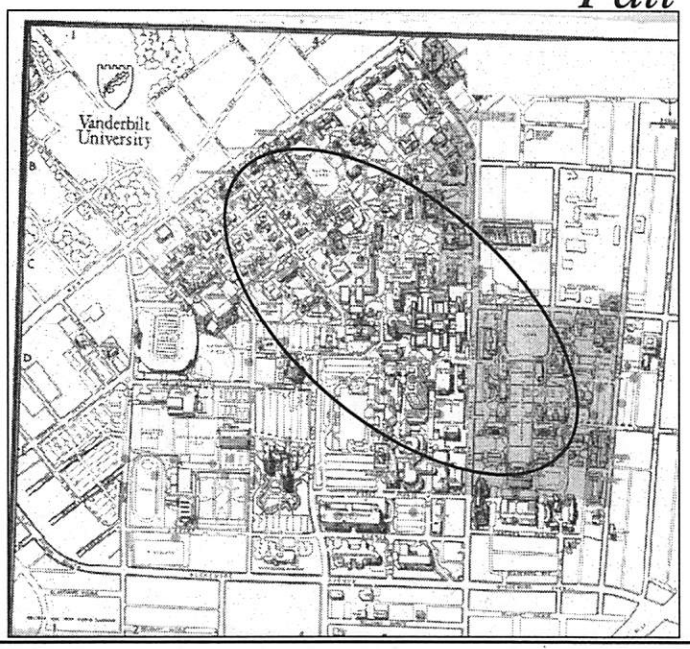

Single Server

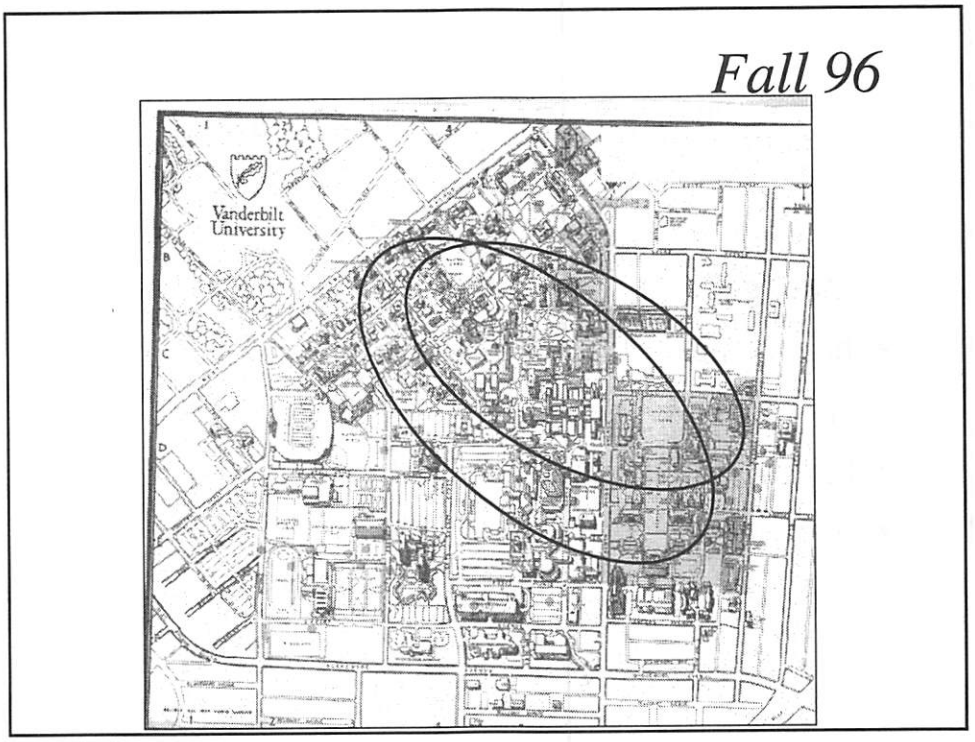

Multiple Server

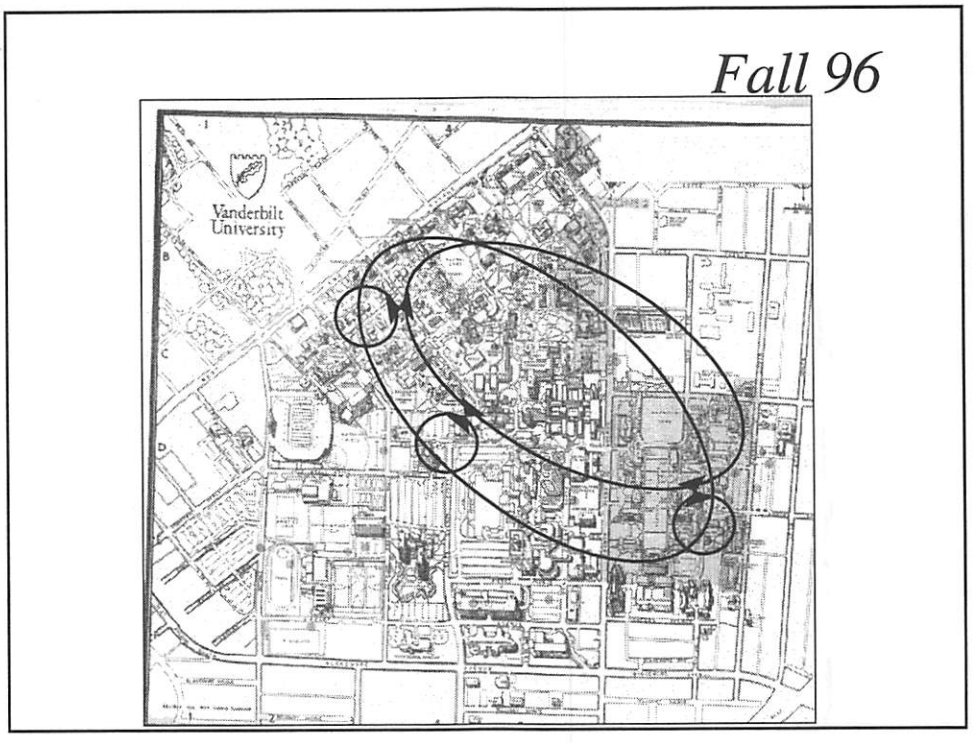
Fall 96

- management
- backbone
- sr
- bryan
- ua

Fall 96




Fall 96
Vanderbilt University

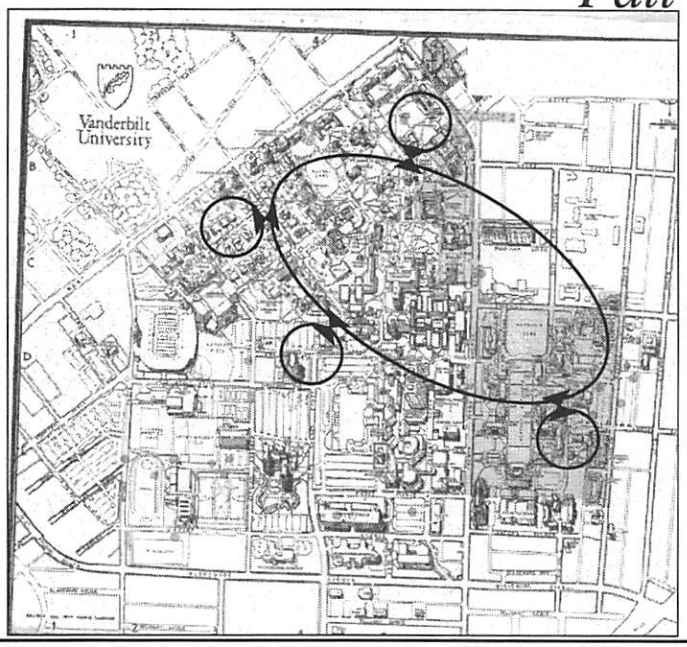

Fall 96
Vanderbilt University

Spring 97

- management
- backbone
- sr
- bryan
- ua

+ wilson

Fall 97



Fall 97

- management
- backbone
- sr
- bryan
- ua
- wilson

+ hill
+ calhoun
+ scieng
+ olin

Spring 98

- management
- backbone
- sr
- bryan
- ua
- wilson
- hill
- calhoun
- scieng
- olin

+ engineering
+ oas
+ video

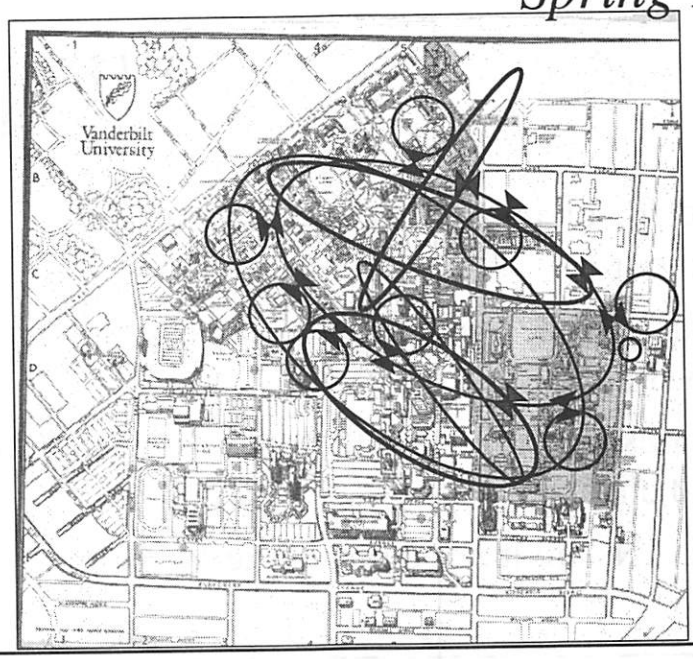
Fall 98

- management
- backbone
+ sr/sr-res
+ bryan/bryan-res
+ ua/ua-res
+ wilson/wilson-res
- hill
- calhoun
- scieng
+ olin/olin-res
- engineering
- oas
- video
+ library
+ telcom/telcom-secure

Spring 99

- management
- backbone
- sr/sr-res
- bryan/bryan-res
- ua/ua-res
- wilson/wilson-res
- hill
- olin/olin-res
- engineering
- oas
- video
- library
- telcom/telcom-secure
+ physics
+ internet/internet-dmz

Spring 99



Future Plans

- Fall 99 - IP routing of 129.59 network
- Spring/Summer/Fall 2000 - MPOA
- Spring 2001 - Layer 3/4 QoS

Summary

- Decide on an architecture, ***then*** choose a vendor
- Make the design as simple as possible, but no simpler
- Choose a framework that facilitates growth and change
- VLANs are just a piece of the puzzle

References

Web Resources

- webster.xylan.com/library/switchbook/05_01.html
- www.interworks.org/conference/IWorks97/sessions/sn084/
- www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/sw_ntman/vlandir/vdir1gsg/overvw.htm
- academy.fore.com/tutorial_index.html
- www.iol.unh.edu/training/atm/thesis/chapter2.html
- www.cavebear.com/CaveBear/Ethernet/index.html

Contact Information

e-mail: john.j.brassil@vanderbilt.edu
phone: 615-322-2496

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of developers, programmers, system administrators, and architects working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering innovation and research that works
- communicating rapidly the results of both research and innovation
- providing a neutral forum for the exercise of critical thought and the airing of technical issues

SAGE, the System Administrators Guild

The System Administrators Guild, a Special Technical Group within the USENIX Association, is dedicated to the recognition and advancement of system administration as a profession. To join SAGE, you must be a member of USENIX.

Member Benefits:

- Free subscription to *;login:*, the Association's magazine, published eight–ten times a year, featuring technical articles, system administration tips and techniques, practical columns on Perl, Java, Tcl/Tk, and C++, book and software reviews, summaries of sessions at USENIX conferences, and Snitch Reports from the USENIX representative and others on various ANSI, IEEE, and ISO standards efforts.
- Access to papers from the USENIX Conferences and Symposia, starting with 1993, via the USENIX Online Library on the World Wide Web.
- Discounts on registration fees for the annual, multi-topic technical conference, the System Administration Conference (LISA), and the various single-topic symposia addressing topics such as object-oriented technologies, security, operating systems, electronic commerce, and NT – as many as twelve technical meetings every year.
- Discounts on the purchase of proceedings and CD-ROMs from USENIX conferences and symposia and other technical publications.
- The right to vote on matters affecting the Association, its bylaws,and election of its directors and officers.
- Discount on BSDI, Inc. products.
- Discount on all publications and software from Prime Time Freeware.
- Savings (10-20%) on selected titles from Academic Press, MIT Press, Morgan Kaufmann, New Riders/Cisco Press/MTP, O'Reilly & Associates, OnWord Press, The Open Group, Prentice Hall, Sage Science Press, and Wiley Computer Publishing.
- Special subscription rate for *The Linux Journal, The Perl Journal, IEEE Concurrency,* and all Sage Science Press journals.

Supporting Members of the USENIX Association:

Apunix Computer Services
Cirrus Technologies
Cisco Systems Inc.
CyberSource Corporation
Deer Run Associates
Hewlett-Packard India Software Operations
Internet Security Systems, Inc.
Microsoft Research
NeoSoft, Inc.
New Riders Press
Nimrod AS
O'Reilly & Associates
Performance Computing
Questra Consulting
Sendmail, Inc.
TeamQuest Corporation
UUNET Technologies, Inc.
Windows NT Systems Magazine
WITSEC, Inc.

Sage Supporting Members:

Atlantic Systems Group
Collective Technologies
Deer Run Associates
D. E. Shaw & Co.
Global Networking & Computing, Inc.
Mentor Graphics Corp.
Microsoft Research
MindSource Software Engineers
New Riders Press
O'Reilly & Associates
Remedy Corporation
SysAdmin Magazine
Taos Mountain
TransQuest Technologies, Inc.

For further information about membership, conferences or publications, contact: USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710 USA. Phone: 510-528-8649. Fax: 510-548-5738. Email: *office@usenix.org.*
URL: *http://www.usenix.org.*

ISBN 1-880446-36-7